Android Malware Classification

A Thesis submitted in the partial fulfillment for the award of

Degree of Master of Technology

In

Computer Science & Engineering

By

Updesh Kumar

(2K15/CSE/20)

Under the Guidance of

Dr. Kapil Sharma

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

Bawana Road, Delhi

CERTIFICATE

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

Bawana Road, Delhi

It is certified that the work contained in this thesis entitled \Android Malware classification ", by Updesh Kumar (Roll No. 2K15/CSE/20), has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Kapil Sharma

Head of Department

Department of Information Technology

Delhi Technological University

Bawana Road, Delhi

July, 2017

DECLARATION

I hereby declare that the Major Project-II work entitled **"Android Malware Classification"** which is being submitted to Delhi Technological University, in partial fulfillment of requirements for the award of the degree of Master of Technology (Computer Science and Engineering) is a bona-fide report of Major Project-II carried out by me. I have not submitted the matter embodied in this dissertation for the award of any other Degree or Diploma.

Updesh Kumar

2K15/CSE/20

ACKNOWLEDGEMENT

First of all, I would like to express my deep sense of respect and gratitude to my project supervisor Dr. Kapil Sharma for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to him for the support, advice and encouragement he provided without which the project could not have been a success.

Secondly, I am grateful to Dr. Rajni Jindal, HOD, Computer Science & Engineering Department, and DTU for her immense support. I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out.

Last but not the least I would like to express sincere gratitude to my parents and friends for constantly encouraging me during the completion of work.

Updesh Kumar

Roll No – 2k15/CSE/20

M. Tech (Computer Science & Engineering)

Delhi Technological University

Abstract

As indicated by AV merchants vindictive programming has been developing exponentially years ago. One of the principle purposes behind these high volumes is that all together to sidestep discovery, malware creators began utilizing polymorphic and transformative procedures. Therefore, conventional mark based ways to deal with recognize malware are being lacking against new malware and the classification of malware tests had turned out to be basic to know the premise of the conduct of malware and to battle back cybercriminals. Amid the most recent decade, arrangements that battle against pernicious programming had started utilizing machine learning approaches. Tragically, there are few open source datasets accessible for the scholarly group. One of the greatest datasets accessible was discharged a year ago in an opposition facilitated on Kaggle with information gave by Microsoft to the Huge Information Trailblazers Social event (Huge 2015). This proposition presents two novel and adaptable methodologies utilizing Neural Systems (NNs) to dole out malware to its comparing family. On one hand, the principal approach makes utilization of CNNs to take in a include pecking order to segregate among tests of malware spoke to as dark scale pictures. Then again, the second approach utilizes the CNN engineering acquainted by Yoon Kim [12] with order malware tests concurring their x86 guidelines. The proposed strategies accomplished a change of 80.86% and 81.56% as for the equivalent likelihood benchmark.

Table of Contents

Table of Figures:

Table of Tables

Chapter 1. Introduction

Malware is nothing but just malicious software. It is intended to spy someone's personal Data without permission of owner. Nowadays due to increase in Smartphone's around the World we need to customize over device's security in order to prevent it from attackers. There is various kind of malware including key loggers, spyware, Trojan horse, virus, worms, Ransom-ware etc. Malware is basically designed for damaging or any illegal action on a system such as collecting sensitive information, to get access control, interrupting CPU operations, advertisement and display that information to distant hacker.

Generally, malware's design is based on its creator intent rather than actual features. Nowadays, Malware creation is on boom due to lure of money in this work through organized internet crime. Nowadays, malware is fabricated to get advantages using advertisement (adware), stealing sensitive information (spyware), email spam or child pornography (zombie computers), to extort money (ransom ware).

1.1 Motivation

Android's open source outline which offer client to introduce unsigned application, permits the clients to introduce applications which are not on the Google's Play [1] or prepared by it. With more than 1.2 million applications accessible for the download from Google's authentic application showcase Google Play, and another 1 million of them spread from the outsider application stores, for example, 9apps and other, as indicated by a gauge there are more than 20,500 new applications which are being discharged each and every month. This requires application store overseers and the malware

analysts to approach the versatile and successful answer for rapidly breaking down new applications to distinguish and segregate them from pernicious applications. Google responds to the expanding enthusiasm of the heels in Android stage by its program called Bouncer in Feb 2012, which is an administration on the Google Play that checks for the malware and alternate perils, when an application that is submitted to the Google Play Store by the designer [10] . Notwithstanding, scientists has demonstrated that Bouncer has the low discovery rate and can be skirted effortlessly [11]. Countless comparative sort of research for the malware of Android has been proposed, however neither of them give a compelling and the far reaching answer for accumulate the intensive comprehension of the obscure and new applications. Blasing et al. constrained their examination to the framework call examination [12,13], Rastogi and Spreitzenbarth track just the particular APIs summons [14,15], and Yan and Yin work is to utilize an emulator or the virtual environment [16]. The Below figure 1.1 shows Android market share worldwide. It also shows frequent users of android.

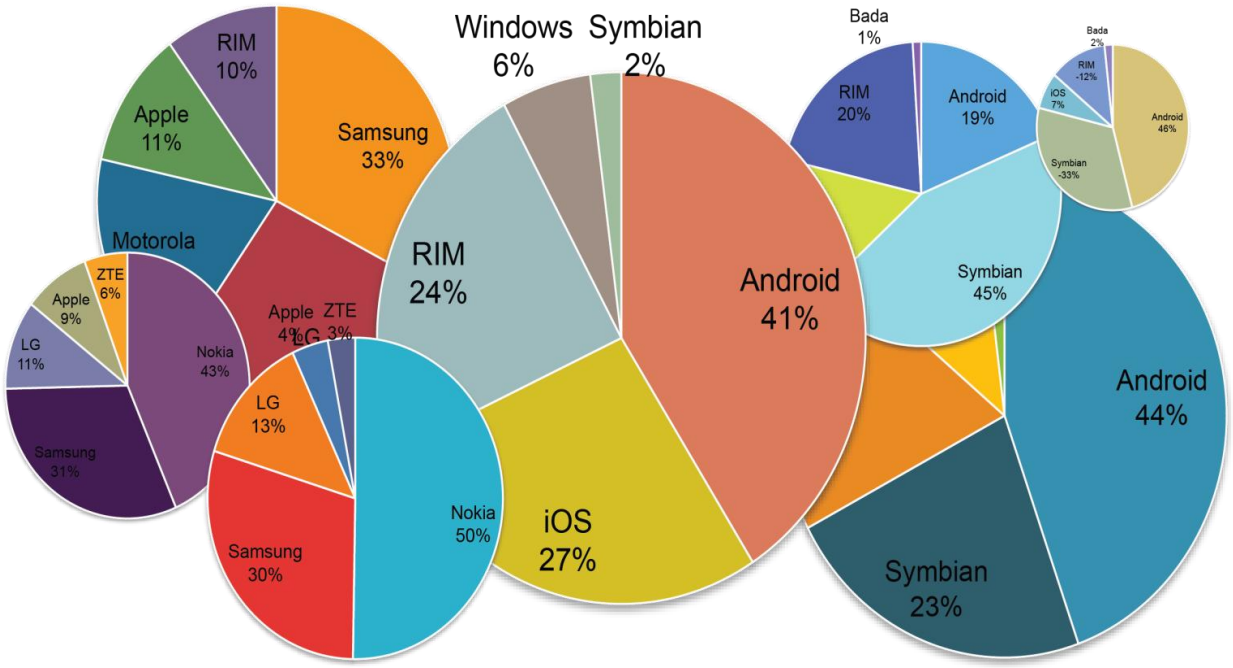


Figure 1.1 Android Market Share

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 technique 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, Hyper Pipes, SMO and Naive Baye's and obtained a accuracy 92.1 %. Shahs 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

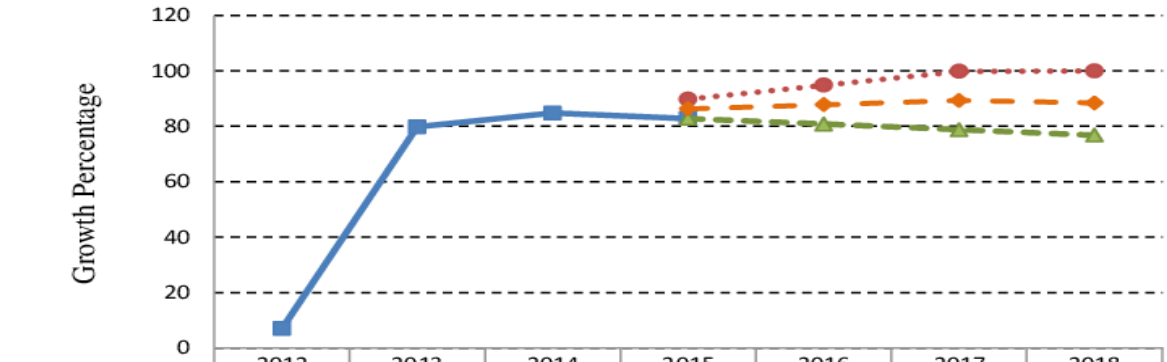
Vector Machine with different Kernels [20] to classify the application whether malicious or not. Peiravian and Zhu [21] use the combination of the permission and API and use the machine learning methods. In their design they extract permission from the app's profile information and the API is extracted 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 current report in the field of utilization has demonstrated that there are close around 2,987,387 Android applications with 12% low quality applications which are right now accessible on the android showcase [22]. The fame of the Android working framework has lead spiking increment in the spread of the Android malware as appeared in Figure 1.2 which illustrates the Android malware development Q1 2017 and expected development in 2017. These malware are chiefly dispersed through the outsider market and stages, for example, 9Apps, however even the Google's play [1] can't guarantee that the greater part of its accessible applications are sans danger. Cases of Android malware incorporate the Phishing Applications, Keeping money Trojans, Spywares, bot, Root Endeavor, SMS Fraud's, Premium Dialer and Fake Installer. Download Trojans are the applications which download their pernicious code after the establishment of it, which implies that this application won't be distinguished by Google innovation amid distribution or transferred by the designer on android showcase Google Play [1]. A large portion of the malware recognition techniques depend on the conventional mark based approaches in which they utilize a database of malware signature definitions, and look at application against this databases of the known malware marks by removing the

arrangement of bytes of code from the application. The significant impediment of this discovery technique is that client is shield from those malware whose current mark has been Transferred to the malware signature database, not from the new malware, zero day assault furthermore, not even from application which encode and transform it code . A past investigation of the malware designs has presumes that the "Mark based methodologies never keep up with the rate at which malware is made and advanced" [23]. This figure 1.3 shows annually growth of malware in android devices .

"**Concentrate highlights from Android applications and utilize them to prepare Neural Systems and its variations for malware identification issue and assess its execution against other machine learning calculations**."



	2012	2013	2014	2015	2016	2017	2018
Current	6.93	79.8	84.8	82.8			
Increase				89.8	94.8	99.8	99.99
Decrease				82.8	80.8	78.8	76.8
Average				86.3	87.8	89.3	88.4

Figure 1.3 Android malware growth percentages

1.4 Objective

The sole point of this proposition is to detect & classify the unknown malware into its respective categories. In this thesis we are using neural network back propagation

algorithm in order to classify the respect malware . The main objective is to analyze the .apk file of android application. The .apk file is passed through different disassemble tools to find its manifest.xml file and source code.dex file, after that we use another tools to parse the manifest.xml file and .dex file. Here we are employing two basic analysis strategies. Static examination and Dynamic investigation, where static examination manages the required consents in show document and touchy Programming interface utilized as a part of that application, though powerful investigation manage the dynamic conduct of use like stream of data, Capacities utilized and Programming interface utilized. To concentrate dynamic conduct we have to run the application in a controlled execution condition.

We have as of now observed many machine learning systems to distinguish and characterize the malware like regression, decision trees, SVM, CNN. We can improve execution in the event that we utilize deep learning notwithstanding neural net strategies. The calculation execution increment up to 96%, much superior to different methods.

1.5 Organization

The theory is sorted out after sections. The first and current part is the presentation, which additionally contains the goals and the association of the postulation. The second section presents the foundation of the venture, concentrating on neural systems and profound gaining from its start as of recently. The third section shows the cutting edge audit with extraordinary consideration on the machine learning calculations and elements used to identify and arrange malware. The fourth chapter contains different Machine learning algorithm basically back propagation to extract the information over

dataset provided by UCI. The chapter Five deals with result over different set of dataset to detect and classify the malware. The chapter Six concludes the proposition and future work to be done.

1.6 Related concepts:

1.6.1 Spyware:

It is type of malware used to spy user information without user permissions. Such as keystroke logger can monitor your activities on internet [1]. Whereas Cookies can track web pages searched. Spyware have ability for Keystroke collection, activity monitoring and financial data gathering.

1.6.2 Virus:

A kind of malicious software having capability of coping itself and advances itself to other systems [2]. It can spread through email attachments. Malicious sites & codes, using infected executables and USB devices. Once Virus is get installed on a system it decrease system performance by curbing .System resources, disable core function & applications and even delete data

1.6.3 Worm:

These are self-replicating Virus reside in active memory of device [3]. They are invisible to user. Worms only experienced when they curb all system Resources & slow down the system. Here the main difference between virus & Worm is that virus need human interface to advancement whereas worm is Self replicating piece of code.

1.6.4 Adware:

It is a type of malicious software often bundled with benign software & Applications to display some kind of advertisement [4]. They are used as a revenue .Generating tool for illegal persons. The ads are delivered by Pop-Up windows or bars that appear on user interface.

1.6.5 Ransom ware:

It is a kind of malicious software [5] which restrict user to access His system by encrypting it data or by locking the system & demand ransom in Order to open it. user 's are forced to pay ransom to gain access .Generally Such Deal occur in bit coin's to remove restricted constraint from system.

1.6.6 Root Kit:

A kind of malicious software [6] designed to control the user system remotely. It controls the remote system without detected by user's or security measures by user. Basically it provides a cover to other malicious code in system being detected by security system.

1.6.7 Back Doors:

A kind of software [8] used to bypass the system authentication & Try to compromise the system. Once system security is compromised in that Case this software automatically creates many backdoors without owner permission & provides many ways to hackers to intrude the system.

1.6.8 Trojan:

It's a kind of software [9] that impersonates itself as normal software. When User downloads Trojan it installs malware on user's system. Trojan horse can be Used to crashing the system , deletion of files, data corruption , blocking the anti-Virus program, formatting disk, spreading malware contents on network, Money . Theft, data theft, spying, surveillance or stalking & use of resources or identity as botnet. Recent example is shedun discovered in 2015 as android malware.

1.6.9 Command & control bot:

Bots are used to automatically perform [11] some specified operations. Bots are responsible for DDoS attack, spam bots that furnish advertisement on web sites. To control these bots the best ways is to Use CAPTCHA tests in order to verify that user accessing the website is not a bot.

Till now, the process of malware detection is done through heuristic based strategies & Signature based approaches. But due to evolution & upcoming new malwares these Strategies are not suitable to detect malware. Signature based techniques usually record the hash of that threat and use that hash value in order to detect upcoming threat.

Most of the malware families belong to certain behavioral families, so it is easier to detect such malware through hash values. Nowadays malware authors became smart enough to escape through signature based approach using polymorphic & transformative malware. Polymorphic malware changes the code whereas the original algorithm remains intact. Some of malware creators use encryption techniques to evade these security measures.

Transformative malware basically transform the written code from one form of representation to another form , for example change the machine language instruction into binary format , once this malware get downloaded at user system then it change the respected binary format to its original form .some of Vendors rely on methods based on heuristics . The process of malware detection is done through the directives directed by experts to detect Unknown malware which mainly depends on dynamic analysis of .apk File of application. Heuristic based approach is much suitable to detect unknown malware but it generates a lot false positive result in compare to signature based techniques. So Nowadays S/W vendor's generally using Hybrid analysis approach to detect unknown malware, before applying signatures they are trying to detect behavior of the malware by running its code and collect information related to Dynamic analysis of malware. So we have two kind of analysis to know a software behavior.

1.6.10 Static Analysis:

It's a technique to analyze the piece of code without executing it. In this technique we usually disassemble the source code and debug the code To extract the useful information such as API calls, required permissions , string Signatures, op code frequency distribution , semantic and syntax analysis of piece Of code , byte sequence etc. In static analysis to analyze the executables, first unpack the file using disassembler and try to find some kind of patterns in order to extract application behavior. In smart phone Era android is emerge as a better platform for different applications. so to detect malware application first disassemble the .apk file to get manifest file.xml and source code.dex file , by using certain tools try to parse the manifest file to get the required permissions in the application . To parse .dex file we

need disassemble to find out the sensitive API's used in the application .So, basically static analysis mainly concentrates over the static behavior of application for android application, we have to check different permissions permitted by platform to user. To detect the malicious software we need to check sensitive API's and permissions, Match these attributes with the recorded signatures/hash values. This process is same as reverse engineering in which an executable (binary format) is passed to disassembler to get original source code. After getting source code user can easily detect the behavior and semantic of intend malware.

1.6.11 Dynamic Analysis:

Dynamic analysis is a procedure to extract useful information from executable by running it. For example in the case of android application to know dynamic behavior we have to run the application in one of the running environment like sandbox, VM, emulators etc. during running of application all the collected information is stored in log for further evaluation . To capture the behavior of application you may use different tools like process monitor, wire shark, process Explorer. This sort of investigation tries to screen capacity and Programming interface calls, the system, the stream of data, and so forth. Contrasted with static investigation, Dynamic examination is more powerful and does not require the executable to be dismantling however it takes additional time and devours a larger number of assets than static examination, being harder proportional. Dynamic analysis expands needs a controlled environment to execute the application which is much differ by genuine application running environment, so in that sense the application may behave different .So due to this analogy, The genuine behavior of malware does not detected because they might need some kind of

triggering mechanism to activate itself like some kind of command, Or some other actions to be happen.

The likelihood of accomplishment of machine learning approaches has expanded because of the conversion of three advancements: (1) the processing control has turned out to be less expensive implying that scientists can fit expansive and that's just the beginning complex models to information and (2) the ascent of business encourages that give volumes of new malware, (3) machine learning as teach has advanced what's more, there are more apparatuses available to them.

Machine learning approaches hold the guarantee that they may accomplish high location rates without the need of human mark era required by customary methodologies. In con succession, organizations and specialists started to utilize machine learning classifiers to help them address this issue.

The principle issue is to see if an android application .apk document is a malware or benevolent programming. So in this proposal we will address two imperative issues

1. Malware detection/ Identification
2. Malware classification/ categorization.

As a matter of first importance we need to address the issue of malware identification, that if a document is a vindictive programming or not. On the off chance that a record falls in the classification of malevolent malware then we need to deliver the issue to characterize the malware into its particular classes in view of its conduct and substance.

Late report demonstrates that almost 6 billion gadgets utilizing android stage, Google play store is the key player here to give diverse application to free. Because of this

openness, these application's are subjected to malware assaults in expansive extent with a specific end goal to stop and identify such assaults we have to devise some security techniques. These days because of advancement of Machine learning approaches it wound up plainly less demanding to develop a model that can without much of a stretch identify and order the obscure malware.

Chapter 2. Background

These days neural system and deep learning strategies is the hotly debated issue in machine learning field and artificial intelligence .Neural net is fundamentally a system of perceptions to mimic the human sensory system . Each perceptron act like a neuron which take some information, applying some calculation over that approaching info and return yield to next neuron in next level. In this part we examine all philosophy behind neural systems.

2.1 Artificial Neural Networks

The starting points of Neural Systems was as calculations that attempt to impersonate the mind and figure 2.1 those a feeling that in the event that we need to fabricate learning frameworks while why not copy maybe the most astounding learning machine we think about, which is maybe the cerebrum. Neural Systems came to be broadly utilized all through the 1980's and 1990's and for different reasons as fame decreased in the late 90's. In any case, more as of late, Neural Systems have had a noteworthy late resurgence. Neural Systems came to be broadly utilized all through the 1980's and 1990's and for different One reason for this resurgence is that Neural Systems are computationally somewhat more costly calculation thus it was just, you know, possibly to some degree all the more as of late that PCs turned out to be sufficiently quick to truly run substantial scale Neural Systems and as a result of that and in addition a couple of other specialized reasons which we'll discuss later, current Neural Systems today are the best in class strategy for some applications.

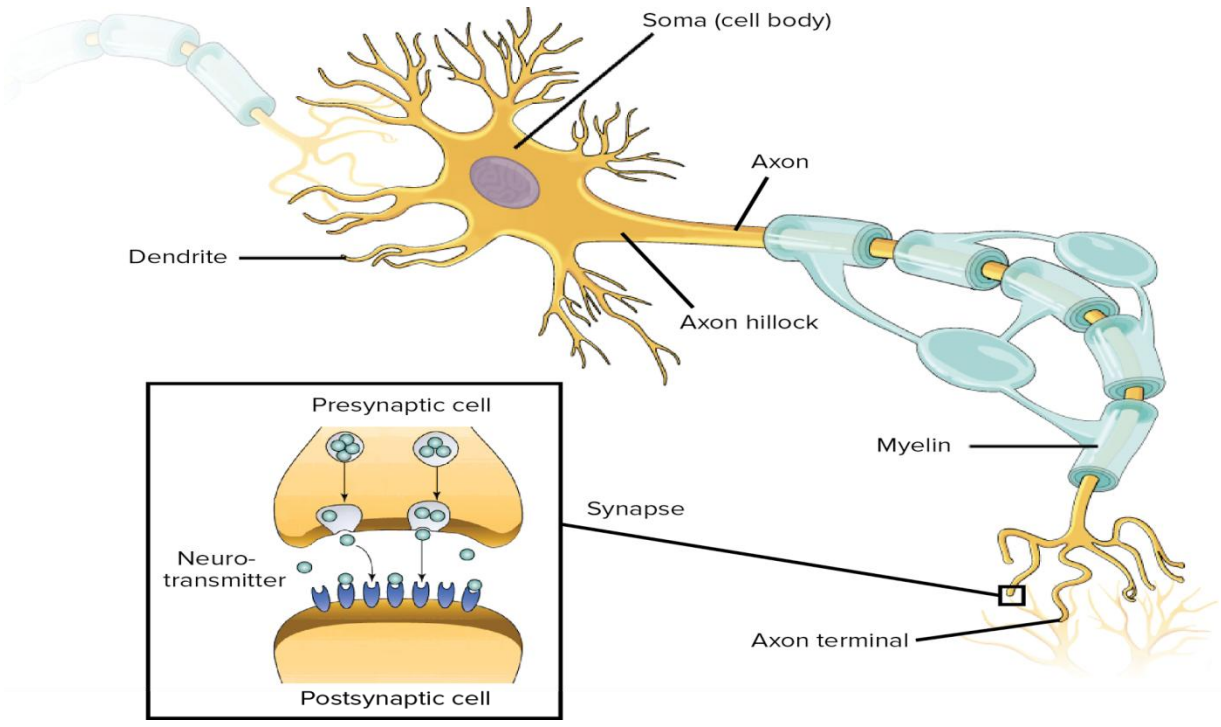


Figure 2.1 Neuron representation

Artificial neural system emulates the conduct of human cerebrum. There is an assortment of neural system frameworks yet here we utilize two frameworks in this recommendation 1. Feed-forward Networks 2. Back propagation algorithm and convolution network (CNN).

This portrayal has no less than three layers of neuron, each layer of neuron have loads of neuron. Every neuron does some calculation from the info and exchange the yield to next layer.

- Neural system framework comprise of info layer and one yield layer.
- Neurons are associated with forward and in reverse layer neuron's to pass data.

- Neurons in same layer are not associated with each other so at same level there is no correspondence and data passing happens.

To comprehend the center idea of simulated neural net, first we require comprehend the key idea of this framework called neurons. How neuron functions, how they pass data amongst forward and in reverse layers.

2.1.1 Perceptrons

Fundamentally, perceptron is a regulated learning calculation at first intended for parallel yield frameworks, whose yield is the 0 or 1. The essential thought behind this calculation is, we have a layered framework have various neurons at beginning and concealed layers yet we just single neuron on the yield layer. Which give the yield as 0 or 1. NNs were first introduced in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt.

This is scientific portrayal of neural model, assume we have a few data sources (x1, x2, x3, x4 ... x N) and having yield (y). To demonstrate the significance of each info Rosenblatt presented the idea of weighted system. So presented weights (w1, w2, w3, w4 ...w N). The yield created by this framework is either 0 or 1. This calculation decide yield, in light of whether the weighted entirety is 0 or 1.

This sort of neural framework is chiefly sent for paired order, Its is not reasonable for multi-class arrangement issue where input has a place with more than maybe a couple classes. To devise this calculation for multi-class order we need to build the quantity of neurons in the yield layer. ANNs are just layers of Perceptrons which take some inputs and produce related outputs.

2.1.2 Simple & Logistic regression

Simple regression is widely used for continuous valued output, where you need to predict values for some inputted data. Suppose you have to find out the price of your house so in that case you need to devise an algorithm that can predict the price of your house. Suppose we have one attribute that is size of house, depend on this attribute we can formulate a function to define the house price sale.

So in view of qualities we have devise a theory that can foresee the house estimation on the off chance that we effectively anticipate the estimations of steady and parameter utilized as a part of this forecast . This speculation can be straight or polynomial in view of the traits of the set. This speculation encourages into decide, that anticipated cost is how much nearer to real cost.

Logistic regression deals with classification problem. Where we need to anticipate the class of information to which it has a place. Assume we have a product and we need to anticipate that, this product is kind or harmful. So we can make two class one with threatening programming and other one to generous.

2.1.3 Sigmoid Function

Perceptrons are constrained in degree. It demonstrates an entire flip flounder in results if any progressions made to weight related with system edges and bias. Perceptrons tuning is truly a troublesome assignment. In the event that some way or another any progressions made to bias related with the layer than yield changes from 0 to 1 and the other way around. So this feature of Perceptrons completely changes the behavior of calculation.

To stop such changes in conduct we present sigmoid neuron. Sigmoid function is genuine esteemed, monotonic and differentiable non negative first derivative. The yield band is lies between 0 and 1. So to take care of classification we utilize sigmoid function. Figure 2.2 shows sigmoid curve graph used in logistic regression .

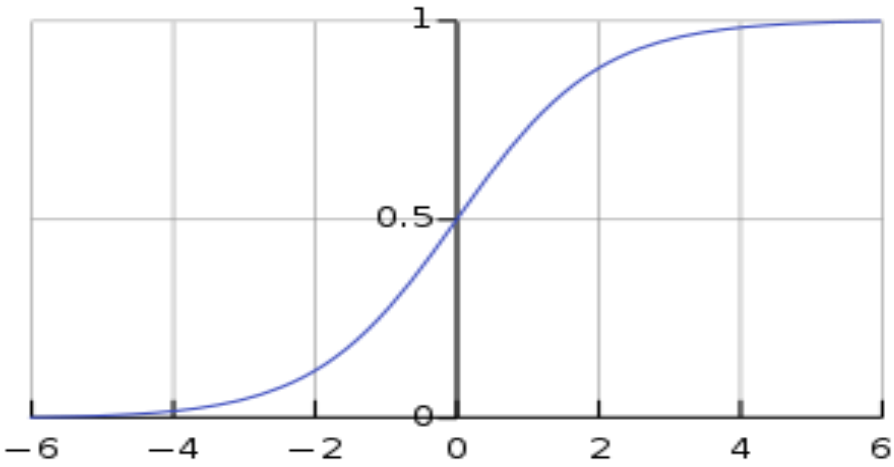


Figure 2.2 Sigmoid Curve

Hence there is only difference in activation function in perceptron and sigmoid neuron.

2.1.4 Loss Function

Loss function or cost function is a measuring apparatus, which demonstrates that how the anticipated yield is far from genuine yield. It demonstrates disparity in yield. So keeping in mind the end goal to compute the inconsistency we utilize MSE (mean squared error).

MSE can be characterized as:

$$\mathbf{T}(\mathbf{S},\mathbf{a}) = \frac{\mathbf{1}}{\boldsymbol{p}} \sum_{\boldsymbol{i}=\mathbf{0}}^{\boldsymbol{p}} (\boldsymbol{h}(\boldsymbol{x}^{\boldsymbol{i}}) - \boldsymbol{y}^{\boldsymbol{i}})$$ **Equation 1**

Where

- p is total number of examples.
- x^i is the i^{th} training example.
- y^i is the actual output for specified set of values.
- $h(x^i)$ is hypothesis value.

So our fundamental goal is to discover the disparity by limiting the squared mean squared error. We need to set the estimations of weights and inclination such that we can limit the MSE. To limit this we will use Gradient Descent algorithm.

2.1.5 Gradient Descent Algorithm

Gradient descent strategy is utilized to limit the difference in cost. This calculation attempt to discover the local minima, to such an extent that around that point we can limit the estimation of weights and bias. Here is stepwise Gradient descent algorithm:

- Right off the bat initializes the weights and bias arbitrarily utilizing randomly initialize. it is utilized to instate haphazardly in light of the fact that in the event that we relegate same function than every concealed layer of NN wind up with an indistinguishable taking in capacity from information .
- Keep iterating the function and update the parameters S,a to reach local minima

$$\boldsymbol{S}_{\boldsymbol{i},\boldsymbol{j}}^{\boldsymbol{l}+\mathbf{1}} = \boldsymbol{S}_{\boldsymbol{i},\boldsymbol{j}}^{\boldsymbol{l}} - \boldsymbol{\alpha} \frac{\boldsymbol{d}}{\boldsymbol{dS}_{\boldsymbol{i},\boldsymbol{j}}^{\boldsymbol{l}}} \boldsymbol{T}(\boldsymbol{S},\boldsymbol{a})$$ **Equation 2**

$$a_i^l = a_i^l - \alpha \frac{d}{da_i^l} T(S, a) \qquad \text{Equation 3}$$

The derivative of this loss function denotes deviation from absolute output. The derivative can be calculated as:

$$\frac{d}{dS_{i,j}^l} T(S, a) = \frac{1}{p} \sum_{i=1}^{p} \frac{d}{dS_{i,j}^l} T(S, a: x^i, y^i) \qquad \text{Equation 4}$$

$$\frac{d}{da_i^l} T(S, a) = \frac{1}{p} \sum_{i=1}^{p} \frac{d}{da_i^l} T(S, a: x^i, y^i) \qquad \text{Equation 5}$$

Here learning rate is essential to choose whether your calculation figures quicker and limit at local/global minima. On the off chance that we pick littler than gradient descent turn out to be slower in calculation however in the event that we pick more than it might avoid the minima and might be diverge.

So here is our principle issue to utilize this calculation is to characterize learning rate such that our calculation additionally works quicker and concurrent at minima. We can pick learning rate by two ways

- Constant decrement/ increment in learning rate at each iteration of gradient descent.
- Gradual decrease in learning rate at each iteration depends on output of layers.

Gradient descents have three variants batch, stochastic and mini-batch gradient descent algorithms.

So we have to control the learning rate in order to classify our inputs correctly .there are many problems faced in gradient descent algorithm , sometimes when we try to fit polynomial curves there exist more than one local minima so in that case our hypothesis

converges at more than one point . So to such problems gradient Descent is not a good algorithm you can use Decision trees to classify and detect a particular pattern in a dataset. For binary classification you can use SVM, it is much better tool than gradient and decision trees in binary classification.

2.1.6 Back propagation

It's a key step behind this thesis working procedure. Back propagation works in two steps (1) feed-forward propagation (2) feed-backward propagation.

In feed-forward stride sources of info are connected at first layer and every node in further layers computes activation function and passes that yield as contribution to next layer . at long last finally layer we figured the contrasts between the genuine yield and the hypothesis value. In encourage in reverse we assess the loss and pass that error to every node in layer course and direct every node to alter their weights and bias as per error.

For a training example (x_i, y_i), apply the feed-forward procedure at each layer and pass the output to next layer (also the output value of the hypothesis $h(x_i)$). So for i^{th} node in l^{th} layer , the error term will be $\delta_i^{\ l}$, it measures the error responsibility of each node in total estimation. For final output layer it is denoted as $\delta_i^{\ n_l}$. For hidden layers the error term denoted as weighted average of the nodes that uses $a_i^{\ l}$ as input.

2.2 Over fitting

Sometimes we have such distribution of data such that each sample is well fitted into the hypothesis. But being as a best fitted model it is not possible to classify a new set of information. So this condition is called as over fitting situation. So here in over fitting

predictive model tries to read noise of data instead of underlying relationship. These models don't yield accurate predictions.

This section deals with the techniques to overcome over fitting problem in large networks.

2.2.1 Regularization

Regularization is the strategy to punish vast weights in systems and like to take in things from smaller weights. To actualize regularization we simply include regularization term in loss function. Regularization does not influence bias added to the system since huge bias make neurons less demanding to saturate which is alluring in specific conditions. Large bias doesn't make neurons more sensitive to data as by large weights.

Regularization has simply two levels:

- L2 regularization

$$\boldsymbol{L(s,a) = L(s,a)_0 + \frac{\lambda}{2n}\sum_s x^2} \qquad \textbf{Equation 6}$$

- L1 regularization

$$\boldsymbol{L(s,a) = L(s,a)_0 + \frac{\lambda}{n}\sum_s |s|} \qquad \textbf{Equation 7}$$

2.2.2 Dropout

The task of prediction averaging is quite tedious for large neural networks. So to address this issue we use Dropout. In Dropout we usually off some of neurons from computation in hidden layer. With the end goal that these neurons can be kept from co-

adjusting each other to much. On each feed-forward cycle, some of neurons present in hidden layer are separated from the system, Now they don't constitute as an individual from system. Separation of neurons happens just in hidden layer not in input and yield layers. So the sustain forward pass connected on changed system and after that data get back engendered on the same adjusted system. On every iteration weights and inclination are refreshed and dropout neurons are reestablished. For next iteration we need to discover the following concealed layer of neurons to be detached on.

As dropout technique in NN's expansion neuron independency on each other so it's a superior thing to learn powerful elements in seclusion rather in conjugation. It appears resembles that on every cycle we are managing diverse neural system. So we need to average every single changed system and after that foresee the yield.

2.2.3 Data Expansion, artificially

To overcome the problem of over fitting, we need to expand the training data to learn more robust features of dataset. It is not a simple task to manually increase the training data. So we can artificially expand the training data. You can increase your data set by cropping, flipping and transposing the images used in datasets.

2.3 Deep Learning

Deep learning is a technique to learn a network having more layers. It is also a part of neural networks terminology. It is same as neural nets but having more number of layers to train the network. So what is the need of deep learning it is same as neural network with more hidden layers, why Deep learning becomes so famous nowadays.

Answers of these questions lies in silent features of deep learning. (1) Computation power needed to train the network. (2) Vanishing Gradient problem

1. Utilization of neural systems to tackle software engineering issues is decade's old. In any case, because of new innovations in scientific equations and increment in calculation energy of PCs Now we can prepare substantial arrangement of information at a quicker pace having bigger arrangement of layers installed in neural nets.
2. The vanishing gradient problem was introduced in [10] by Sepp Hochreiter,as chain rule is applied to compute the back propagation, where the actual computation is done by activation functions like sigmoid function or hyperbolic tangent where gradient range lies in [-1,1] or [0,1] . So if we have a N layer neural nets, on multiplying the outputs of the yield layer with N, results that gradient decrease exponentially with N. So the training of front layers is much slower than other layers.

Figure 2.3 shows that if we have high learning rate than loss function will diverge whereas on good learning rate it will converge and having low epoch.

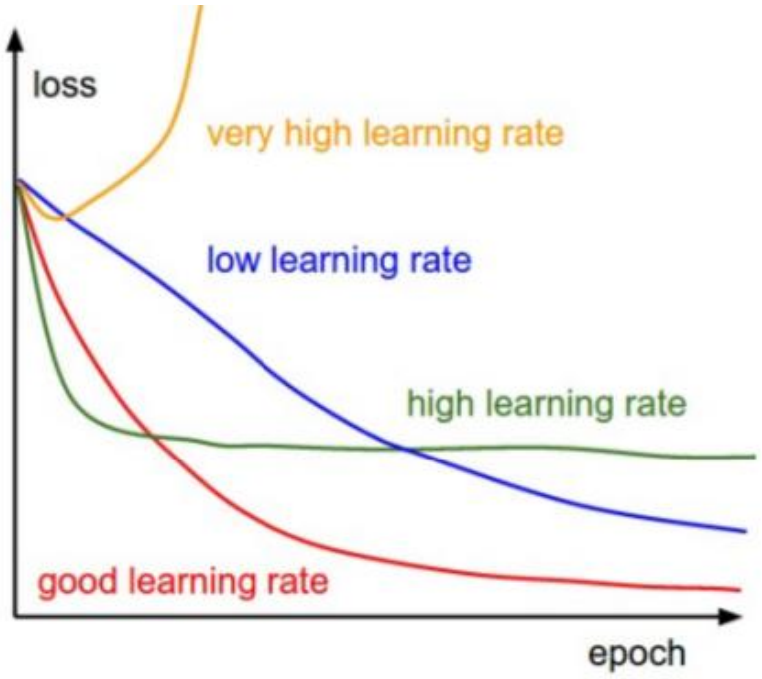


Figure 2.3 Effects of different Learning Rates

2.3.1 ReLU Units

To solve the vanishing Gradient problems we are using rectified linear units (ReLU) introduced in 2010. Until then this problem is not solved. The activation function for ReLU units is defined as:

$$f(x) = \sum_{j=1}^{\infty} \sigma(x - i + 0.5) \approx \log(1 + e^{x}) \qquad \text{Equation 8}$$

Where

- $\sum_{j=1}^{\infty} \sigma(x - i + 0.5)$ Is stepped sigmoid function.
- $\text{Log}(1+e^{x})$ is softplus function.

We can approximate this soft plus function soft plus function to **max function or hard-max** function as max(0, x+N(0,1)). It is also called as rectified linear function (**REL**).

So to stop the vanishing gradient problem we are using REL. REL is differ from sigmoid function in many aspects. Some of aspects are defined below.

1. Sigmoid capacity is utilized to display the likelihood as its range lies between [0,1] .though REL is intended to anticipate or display genuine esteemed numbers as its extents lies in range [0,] .
2. Angle of sigmoid capacity vanishes as we increment or lessening estimation of x though slope of REL doesn't vanish on expanding or diminishing the x esteem.
3. In the event that we utilize hard-max work as enactment capacity then it will actuate sparsely in shrouded layers.
4. REL doesn't require pre-preparing to prepare the neural systems.
5. ReLU can be utilized as a part of Limited Boltzmann machine to model genuine/whole number esteemed information sources.

There are various variants of ReLU namely noisy, leaky and ELUs. There are many problems associated with ReLU. Some of them are listed below:

1. This activation function is non differentiable at zero. It is differentiable anyplace, and furthermore differentiable near zero however not at equivalent to zero.
2. It is non-zero focused.
3. It could possibly explode in light of the fact that it is unbounded.
4. ***Dying ReLU problem*** : ReLU neurons here and there winds up noticeably dormant with the end goal that they don't pass data and reach in idle state to all sources of info and in reverse yields . These neurons are called as biting the dust or dead neurons. We have to separate these neurons. This framework

diminishes the model limit and this impact can be relieved by utilizing leaky ReLUs.

With the help of ReLU we have a lot of algorithm that provides optimum utilization of gradient descent algorithm. ReLU provides faster and effective learning for deep neural networks in comparison to sigmoid function on asymmetric and complex dataset.

2.3.2 Gradient Descent optimization Algorithms

There are three variants of gradient descent algorithm which can be used to optimize the basic gradient descent algorithm.

- ***Batch Gradient Descent Algorithm:*** This algorithm calculate the gradient over the loss function T(S,a) for the entire training set in one go. It gives ensured convergence to convex surfaces over global minimum and local minimum for non raised surfaces.
- ***Stochastic Gradient descent:*** it is an incremental gradient descent which refreshes its parameters iteration by iteration .it is utilized to limit the target work that composed as total of differentiable capacities. This calculation tries to discover maxima or minima by iteration. This algorithm can be explained as: suppose we have an gradient to minimize $w=w-Q_i(w)$. So to minimize iterate it through a loop up to the function is not minimized.
- ***Mini-Batch Gradient Descent Algorithm:*** this algorithm appears to be same as batch gradient descent as the name suggest. Here we are trying to build up a batch of n (smaller set of total samples) training samples to learn the model. So the gradient will be calculated over the loss function of n training samples. This

algorithm have better convergence rate than stochastic and batch gradient descent algorithm.

Next it is displayed a framework of a few calculations utilized as a part of profound figuring out how to upgrade the gradient descent algorithm.

2.4 **Momentum**

The basic gradient algorithm may have slow learning rate or faster rate, in figure 2.4 either case algorithm does not converge perfectly. Steepest gradient descents modify weights and bias as:

$$S_{i,j}^{l} = S_{i,j}^{l} - \alpha \frac{d}{ds_{i,j}^{l}} T(S, a) \quad \text{Equation 9}$$

$$a_{i}^{l} = a_{i}^{l} - \alpha \frac{d}{da_{i}^{l}} T(S, a) \quad \text{Equation 10}$$

Here learning rate can be very slow, so the rate of convergence become very slow . To increase this convergence rate we add momentum term to basic gradient algorithm at each time step t.

$$S_{i,j,t+1}^{l} = S_{i,j,t}^{l} - \alpha \frac{d}{dS_{i,j}^{l}} T(S, a) + \gamma S_{i,j,t-1}^{l} \quad \text{Equation 11}$$

$$a_{i,t+1}^{l} = a_{i,t}^{l} - \alpha \frac{d}{da_{i}^{l}} T(S, a) + \gamma a_{i,t-1}^{l} \quad \text{Equation 12}$$

Where γ is the momentum term. So the modification in weight vector takes place according to gradient term of this time step and weight vector of previous time step t-1.

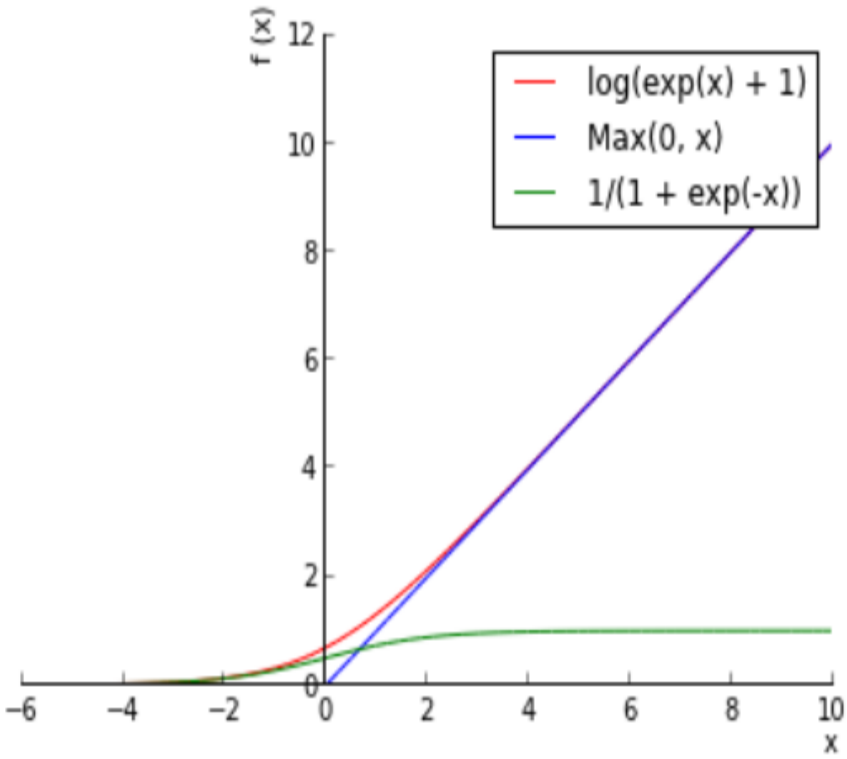


Figure 2.4 REL and Sigmoid Function Comparison

Figure 2.4 Here shows the graphical comparison between ReLU and sigmoid function.

2.5 ADAGRAD:

This algorithm used to optimize the learning rate of gradient descent algorithm. Its an adaptive approach and used to update the learning rate on the basis that how this algorithm behaves on previous iterations. So on basis of previous iteration data learning rate of gradient descent algorithm can be updated.

At each time step t this algorithm updates and adjust the weights of networks or you may say it will adjust parameters $S_{i,j,t}$ ^{l}. it will update learning rate on each parameters and it will perform smaller updates for frequent parameters whereas infrequent parameters have larger updates based on the past gradient values .

$$S^{l}_{i,j,t+1} = S^{l}_{i,j,t} - \frac{\alpha}{G^{l}_{i,j,t}+\epsilon} * \frac{d}{dS^{l}_{i,j,t}} T(S, a)$$ Equation 13

2.6 ADAM:

It is also one of the adaptive techniques to optimize the gradient descent algorithm. The strategy is clear to actualize, is computationally effective, has little memory prerequisites, is invariant to corner to corner rescaling of the angles, and is appropriate for issues that are extensive as far as information or potentially parameters. a calculation for first-arrange inclination based advancement of stochastic target capacities, in view of versatile assessments of lower-request minutes. The technique is likewise fitting for non-stationary destinations and issues with exceptionally uproarious as well as inadequate angles.

The hyper-parameters have instinctive elucidations and commonly require small tuning. A few associations with related calculations, on which Adam was propelled, are talked about. We likewise break down the hypothetical merging properties of the calculation and give a lament bound on the joining rate that is equivalent to the best known outcomes under the online irably by and by and looks at positively to other stochastic advancement strategies. At last, we examine AdaMax, a variation of Adam in view of the limitlessness standard. Arched streamlining system observational outcomes show that Adam functions. It stores an exponentially decaying average of the past squared gradients that we will denote v_t and similar to momentum, it keeps an exponentially decaying average of past gradients m_t:

$$m_t = \beta_1 m_{t-1} + g_t v_t * (1 - \beta_1) = (1 - \beta_2) * g_t^2$$ Equation 14

What's more, g_t is the angle of the target capacity and β1 and β2 are the rot rates. M_t and v_t are the evaluations of the main minute or mean and the second minute or un centered difference of the inclinations, separately. M_t and v_t are instated as vectors of 0's and in consequence, during beginning time steps and when the rot rates are little they have a tendency to be one-sided towards 0. To take care of this issue, they figured the predisposition remedied first and second minute appraisals:

$$\boldsymbol{m_t = \frac{m_t}{1 - v_t}} \qquad \textbf{Equation 15}$$

As a result, The Adam update rule is defined as:

$$\boldsymbol{m_t = m_t - \frac{\alpha}{\sqrt{v_t} + \in} * m_t} \qquad \textbf{Equation 16}$$

So these are certain tricks to optimize the basic gradient descent algorithm

Chapter 3. Literature Review

This section will give a short depiction of the key ideas and the phrasing identified with the Android Operating System Architecture, Android application structure, Linux framework calls, the parts of the Android applications and the building squares structure, for example, exercises, the administrations, recipients and the goals of the Android applications. At last about the malware what are it, its different sorts and how it takes preferred standpoint of the android stage?

3.1 Android System Architecture

The product pile of the Android stage is appeared in Figure 2.1. The violet things are the libraries and parts which are local code(C/C++), green things in the Java dialect parts which are executed and translated by the Dalvik Virtual Machine [28].The last red base layer speak to the parts of Linux portion and execute in the portion space of the Android working framework. In the accompanying subsection, we will talk about the diverse reflection layer of the android framework engineering. For the more detail review of it, we allude to the prior examinations [28].

3.1.1 Linux kernel

Android utilize a unique variant of the Linux's Kernel composed by the Linus Torvolds with the uncommon augmentations to adapt to the different necessity of the installed framework. figure 3.1 incorporate wake locks i.e. it is the systems which demonstrate that the application need that the screen should continue showing and does not goes off, the Binder IPC driver, a memory management framework more stubborn in

protecting memory, and different other highlights which are critical for the versatile installed stage having low equipment assets.

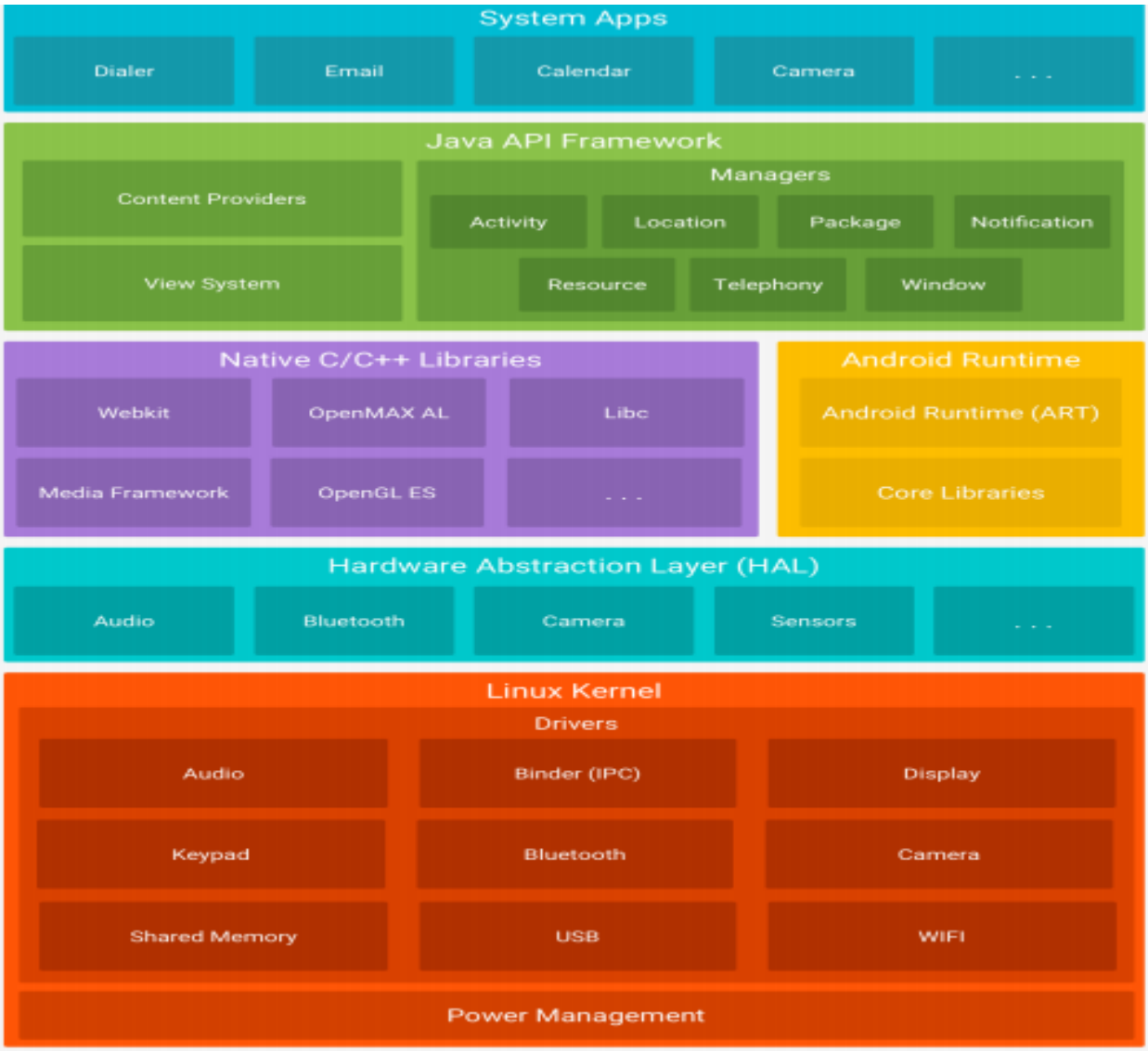


Figure 3.1 Android Platform Low Level System Architecture

3.1.2 Libraries

Application Framework contains set of the local libraries which are composed in C/C++ and these libraries are likewise accessible to the Android Runtime by the Libraries

Component. These are by and large open source outside libraries, for example, OpenSSL, WebKit and bzip2 with just exceptionally minor alterations. The basic C/C++ libraries codename as Bionic were gotten from the BSD's libc and were composed again for to help the ARM equipment direction set and execution of the p-strings in light of Linux futexes ("quick client space mutex") by Android.

3.1.3 Android runtime

The Component in the middleware of the android framework is called Android Runtime which comprises of the Dalvik Virtual Machine (DVM)[29] and had the arrangement of the Core Libraries. Dalvik Virtual Machine resembles the JVM which is in charge of elucidation and execution of the applications that are composed in the Java programming dialect and talked about in detail in Section 2.2. The center libraries establish the framework for the usage of the broadly useful API for the applications that are executed by Dalvik Virtual Machine. Android recognizes the two classifications of the center libraries. 1. Dalvik Virtual Machine indicated libraries. 2. Java or Java driven interoperability libraries. The first permit in changing or handling Virtual Machine particular data and is for the most part utilized when guidelines or byte codes are should be stacked on the memory. The second one gives the cordial condition to the Java software engineers and is gotten from the Apaches Harmony. It executes the greater part of the mainstream Java dialect bundles like java.lang and java.util.

3.1.4 Application framework

The Application Framework of the android stage gives building squares of abnormal state or the deliberation to the applications in type of different bundles of android. The vast majority of these segments in this layer are executed as the applications which

keep running out of sight as the foundation forms on the cell phone. Different segments are in charge of the administration of fundamental telephone usefulness like accepting telephone calls, instant messages or checking power use and so on. A few segments which are more essential are, for example. A couple of components which are more important are such as:

3.1.4.1 Activity Manager

The Activity Manager is prepare, which keeps the track applications which are as of now dynamic. It slaughters the foundation forms if the gadget is out of memory. It additionally had the ability for identification of the non-responsive applications i.e. at the point when an application does not offer reaction to the info occasion in 5-10 seconds (like a key squeezed or screen is touch). It subsequently prompts an exchange for the Application Not Responding.

3.1.4.2 Content provider

These goes about as the essential building obstructs for the android application. They can share the information between the different various applications. Address book information, for instance, should be gotten to by the diverse applications and in this manner it is put away in the substance supplier.

3.1.4.3 Telephony manager

The Telephony Manager gives the entrance to the communication administrations' data of the gadget, for example, the telephone's one of a kind gadget ID number (IMEI), the

cell current area or serial number. It is likewise in charge of administration of the telephone calls.

3.1.4.4 Location manager

The Location Manager give the framework area administrations access to the android application which enable the applications to get updates of gadget's land area occasionally by the assistance of gadget's GPS sensor.

3.1.5 Applications

Applications are in charge of the connection amongst gadget and client, and are based upon the highest point of the Application Framework. It is impossible that the normal client of the android ever need to manage the segments are not in this layer. Pre-introduced applications which accompany packaged android framework offer various fundamental errands to the client jump at the chance to perform like surfing web, perusing content, making telephone calls and so on., however client is allowed to introduce the outsider applications to utilize different elements like (e.g., gaming, watching recordings, perusing news, route through GPS, perusing books, and so forth.).

3.2 Dalvik Virtual Machine

The Dalvik VM [29], resembles a Java Virtual Machine uncommonly composed and adjusted for memory enhancement and diminish vitality utilization in the inserted frameworks like advanced mobile phones, tablets and brilliant TVs. It was composed and made by the Dan Bornstein, with joint effort and commitment with Google engineers. This virtual machine is enhanced to requires a low level of the memory use and empowers different. The Dalvik VM utilizes enroll based engineering [30] clarified in

Section 2.2.3, which is quicker and more effective than the stack-based design [31] utilized as a part of other virtual machines. Each application in the android framework runs its own particular procedure, with its own special occasion of the Dalvik Virtual Machine inside the protected condition, called Sandbox [15]. The Dalvik Virtual Machine executes records which are in the Dalvik VM executable arrangement (DEX Format) appeared in Figure 2.2, which is the streamlined variant of the Java code petition for the frameworks with obliged memory and moderate processor speeds.

3.2.1 Hardware Constraints:

The Android stage is particularly intended to work on the cell phone and in this manner it needs to come over the testing equipment confinements which when contrasted with the normal PC working frameworks condition, the cell phones are little in the size and fueled by the little source for the most part a battery. Because of this, the underlying cell phones contained a moderate CPU generally and had next to no measure of the memory left once the framework is booted.

In spite of these antiquated and low details, the Android working framework relies on the present day Operating System standards: every application should keep running by making it possess prepare and has the memory space of its own which implies that every application on android should keep running in its own particular virtual machine. It is contended that the security necessities won't be happy with these restricted equipment requirements with the utilization of the current Java virtual machines which is ported to the android framework. To defeat this, Android had utilized the Dalvik Virtual Machine. An uncommon example of the preloaded framework classes. Besides, restricted to the Stack-based virtual machines an instrument which can be ported to the

any stage of the Dalvik Virtual Machine is Register-based virtual machine and is intended to keep running on the ARM processors particularly. This enables the Virtual Machines engineers to include more speedup advancements.

3.2.2 Byte-code

The byte-code is translated by the Dalvik Virtual Machine thusly it is called DEX byte-code or Dalvik Executable Code. Figure 3.2 demonstrates that the DEX code is acquired by transformation of the Java dialect byte code utilizing the DX apparatus. One of the principle contrasts between the Java dialect byte code and DEX document organize is that all of code is repacked into single yield record (classes.dex), and copy capacities, strings and code pieces are expelled. Actually, this outcomes in the more utilization of the pointers inside the DEX byte code itself than in Java .class records. As a rule, .dex records are around 6% littler than the packed .jolt documents. Amid the Android application establishment, classes.dex document which is incorporated with the application is enhanced and confirmed by the android working framework. Confirmation is done so as to ensure that the program must act ordinarily by decreasing the runtime bugs. Streamlining includes in covering of exceptional strategies, static connecting and pruning vacant techniques.

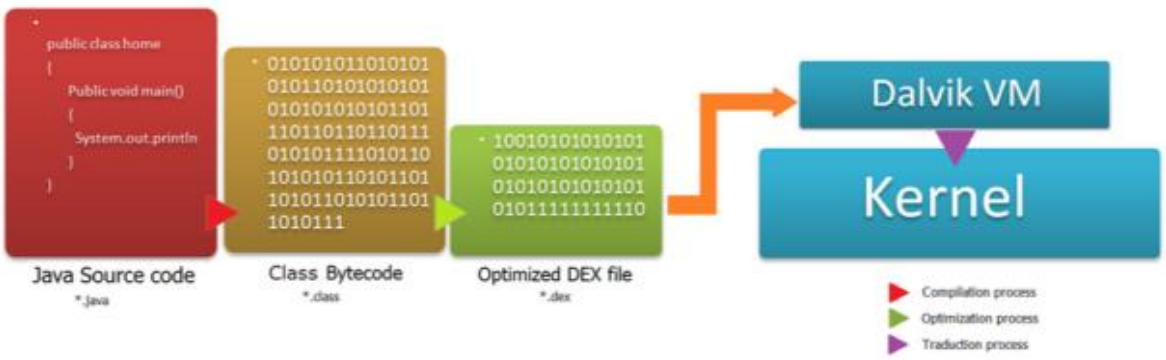


Figure 3.2 DEX file creation

3.2.3 Register-based Architecture

Virtual machine engineers have dependably been in the contention of executing virtual machines with stack-based design [31] instead of enrolled based engineering [30]. The extremely straightforward usage of the stack-based design drives the engineers to incline toward it utilize. Clearly, this basic usage of the stack-based machine accompanies execution cost. Executables for stack-based design are compacted than executables for enroll based engineering because of more utilization of the POP and PUSH operations. Which prompts higher memory utilization, prompting the more terrible execution of the virtual machine? Enroll based engineering for the most part requires a normal of 48% less executed virtual machine directions than the Stack based design, which impressively enhances execution of the gadget. While then again, the enroll code utilized by enlist based engineering is bigger than stack-based design code. Despite the fact that, the handling load produced by the Register-based design is still lower than that of the Stack-based engineering. Considering the way that the Dalvik Virtual

Machine runs easily on the inserted gadgets with compelled memory and handling power, the utilization of the enlist based design is the a great deal more proper decision.

3.2.4 Android Application

The android applications are coded in the Java programming dialect. Android utilizes Java's customizing IDE and Android's Software Development Kit (SDK, for example, Android Studio [32], to make an Android application establishment APK record by the accumulation Java code. These APK records would then be able to be later introduce on Android gadgets by the Android Debug Bridge device called ADB or from the Google's Play. figure 3.3 demonstrates the fundamental structure of an APK record.

Figure 3.3 APK File Composition

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

3.2.4.1 AndroidManifest.xml file

The Android show record contains the most essential data of the Android application. It portrays the application components, for example, the bundle name, application name

and the authorizations utilized by the application also the base adaptation of the Android Operating System which is required by the application to run.

3.2.4.2 Classes.dex file

This document is yield of aggregation of source code in the Android Java dialect. It contains advanced .DEX Byte code of Android application and keeps running on the Dalvik Virtual Machine.

3.2.4.3 Resources:

This gathering contains the photos, media, libraries and the format documents utilized by android application. A standout amongst the most vital components of making an APK record is the aggregation of the Java source code. The way toward creating the APK record is portrayed in Figure 3.4. The documents experiences to a progression of changes amid the procedure of the making of the Android APK record. The changes contain the aggregation procedure required to the create APK documents that will keep running on the Android gadgets.

The initial step of procedure of the creation an Android application is to make an Android extend, in which Java source code, Android show and asset documents will be produced by Android Studio. The following stage is to program and design the code to suit the reason and to assemble the venture. The Java's compiler in the SDK programming condition will create the class records from the Java's source code and the AAPT will change the AndroidManifest.xml and asset documents into the satisfactory arrangement with the goal that they can be translated by the Dalvik Virtual Machine. The created classes' documents can't be deciphered by the Dalvik Virtual Mc.

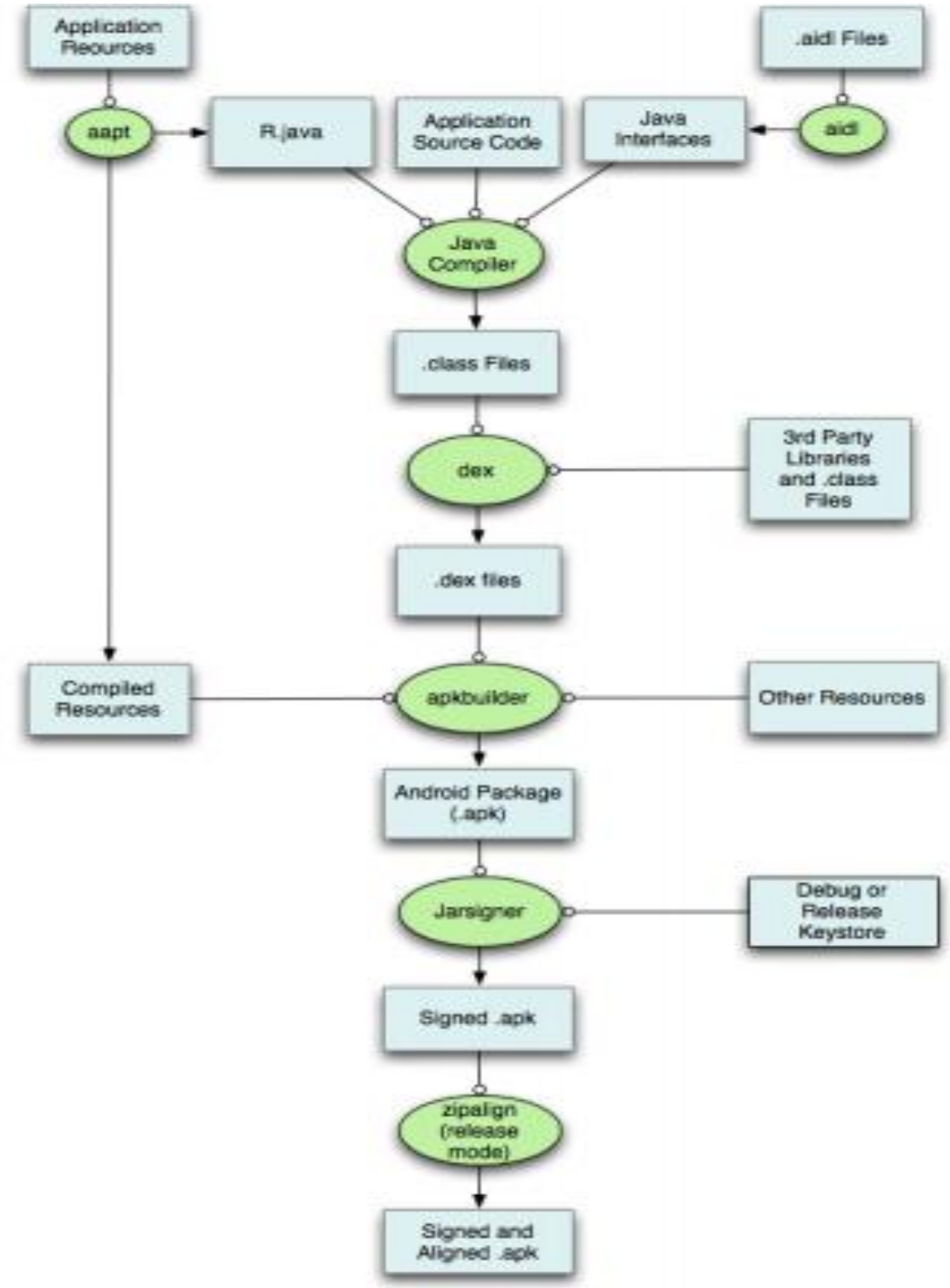


Figure 3.4 APK file build process

3.2.5 Application components

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

3.2.5.1 Activities:

It speaks to the single screen with the specific interface for the client. Applications can have any number of the exercises for the diverse reason. Like a web program, for an example, may have one of the exercises to demonstrate the rundown of history things while another movement to deal with the setting of the web program. Every movement is autonomous of the other action and can be begun by alternate applications. An email application, for instance, can begin the web program action to see of got hyperlink.

3.2.5.2 Services:

Services' are the parts which rushed to play out the long running operations while running in foundation and don't have an interface for the client. The email application, for instance, will have an attachment benefit which is in charge of downloading messages from the server out of sight at the same time the client is cooperating with the diverse application. Administrations can be begun by alternate segments of uses like action or by the communicate collector. Content Providers give access of the information between various utilizations of the android applications. They use to oversee shared set containing the application information. Update Information, for instance, is put away inside the substance supplier and other application can get to it when required.

3.2.5.3 Broadcast receivers:

It tunes in to the specific communicate declarations from the framework and respond on these. By and large the vast majority of the communicate are from the framework and it gets them and reports them. They don't have a UI and go about as the door to send

data to alternate parts of the android framework. For instance if battery is low then framework send the communicate and after that the communicate collector will declare it and the movement will catch it and alarm the client by showing a ready box on the screen.

3.2.6 Distribution

Android clients can introduce applications it is possible that it is from Google or its from different engineers or by means of the Google Play. Google Play is a validated online conveyance stage of utilization of Google from which client can download the free and paid application from the designer. Google has assemble an in-house completely robotized antivirus framework called Google Bouncer which examine the transferred application and checker whether they may not contain malignant code. Clients can introduce the application from alternate sources than the Google Play, for this establishment the client must empower the permit establishment from the obscure source choice in the settings. By this client can introduce the outside APK records downloaded from the web and additionally the outsider markets. The outsider markets like 9apps, Mikandi's application store, and Xiaomi application store give client particular offers.

Chapter 4. State of Art

Before the machine learning, it's quite tedious to detect and classify the malwares in android and non-android devices. from a decade ago scientists and hostile to infection merchants begins utilizing the machine learning calculations like SVM, decision trees , association rules , naive baye's , random forest , neural networks and now deep learning to detect and classify different malwares . So in this section we are going to discuss all the major approaches used in literature.

4.1 Byte-sequence N-grams

N- Grams based algorithm used as a classifier firstly in IBM's antivirus scanner. It is suitable for the analysis of files having their description in hex values. Figure 4.1 N gram is just a sequence of N hexadecimal values given in a malware file. It may take up to 257 values , having range up to 256 and one special value '??' . Researchers may take 3-grams as features and neural network as classification model. The size of features increases exponentially as we increase value of N. for example if we take bigram than total number of features will be 2562 and for trigram features increase to 2563. So we have to use some approach to decrease these features such that it does not decrease the model capacity.

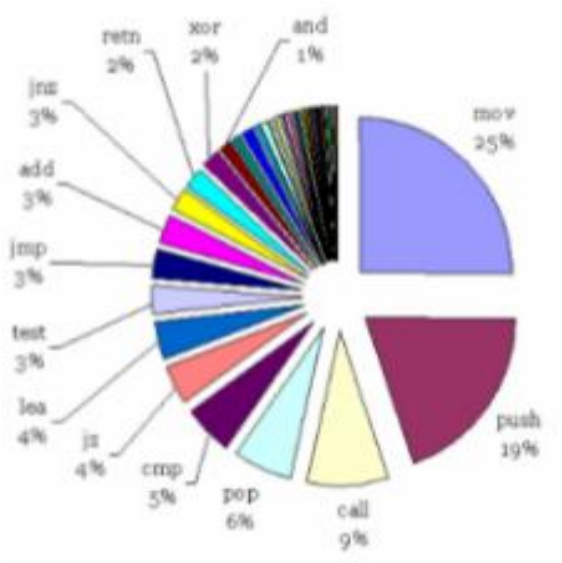


Figure 4.1 most frequent 14 op codes in Good-ware

4.2 Op code N-grams

This method is suitable to analyze the assembly code of malware files. Resembling to the byte sequence N-gram the model is prepared using the op codes extracted from assembly language code files. As op codes represent the type of operation CPU is going to perform in current machine cycle. Such as memory read, write or any other operation performed by processor. So software may have both frequent and rare op codes. This software act as malware/good ware. The following below charts represent the 14 op codes for good-ware and malware respectively those are most frequent nowadays. With a specific end goal to diminish the quantity of Op Code n-gram features, which ranges from thousands to millions, we utilized the DF measure to choose the main 1,000 components and tried three element selection techniques. The 2-gram figure 4.2 Op Codes beat the others and the DF was the best component selection technique. We additionally assessed the execution of classifiers when utilizing

a steady size of Op Code n-grams as opposed to utilizing differing sizes of n-grams. Here are charts of op codes in good-ware and malware.

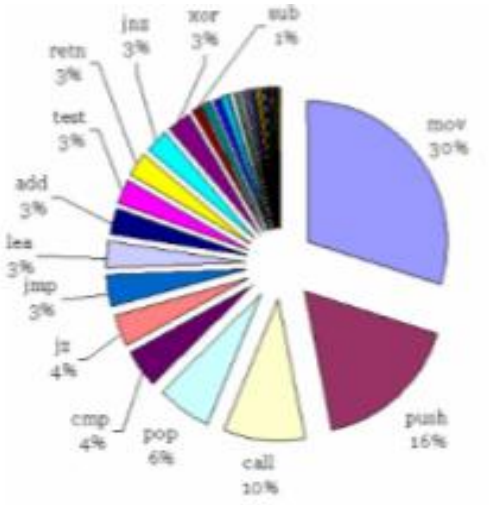


Figure 4.2 Most frequent 14 op codes in malware

4.3 Portable Executables

PE arrangement is fundamentally used to speak to the executable records, DLLs. it likewise give a superior information portrayal that exemplifies the required data for the administration of executables by working framework loader. It might incorporate dynamic connecting libraries and distinctive Programming interfaces to bolster import and fare of information. So to remove the data from the executables we have to perform static analysis on the structure of PE. This auxiliary data demonstrates that whether executables is manipulated or distorted by any malicious movement .there are some set of features that can be extracted from the PE's.

1. Analysis of stored file pointers on disk.

2. Import sections which describe functions from which DLLs were used and the list of DLLs of the executable that are imported.
3. Function exported and their exported sections.
4. PE header structure and having such features like the code and file size, the creation time, etc.

There are many frameworks to work on in order to extract the static information from PE's like Invencea. Invencea labs modeled a deep learning approach to classify and detect the malware based on the structural analysis of PE's .The deep NN comprised of three layers: the information layer of 1024 info components and figure 4.3 two hidden layers of 1024 PReLU units every one. The yield layer comprised of one sigmoid unit indicating whether the record is good-ware or malware. In the accompanying picture you can see a diagram of the system.

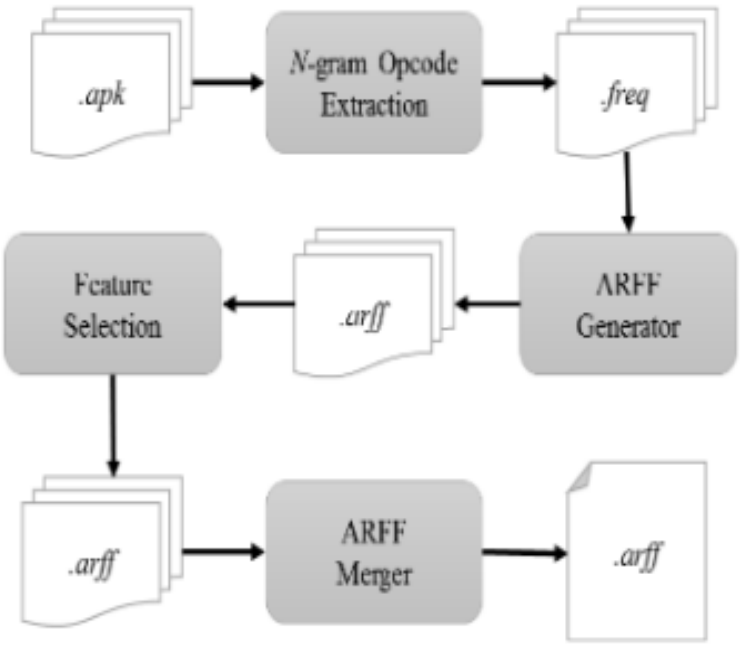


Figure 4.3 Flowchart of feature Extraction

4.4 Entropy

It's a tool to analyze the entropy of a PE section of a executable file in order to detect that whether the file is encrypted or not. So the standard calculated average entropy is 4.347, 5.099, 6.801, and 7.175 for plain files, native executables, packed executables and encrypted executables, respectively. figure 4.4 Shows that Due to introduction of signature based techniques now malware authors stated to use obfuscation techniques to pass through anti-virus systems. So here is the need to find out the changes in the statistical nature of malware files. Here entropy plays a main role in order to detect such features.

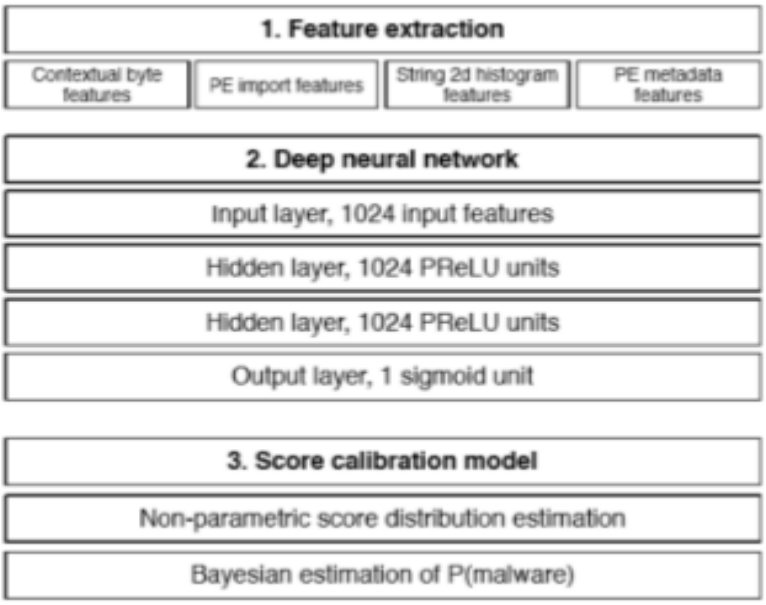


Figure 4.4 Invencea's malware detection framework Outline

4.5 API calls

Application Programming interface calls utilized by android application used to get to various utilities can be analyzed by static examination. For removing the Programming interface utilized as a part of .apk record first we need to dismantle the .apk document

utilizing distinctive disassemble today's. Initially uncompress the .apk record utilizing 7-zip apparatus and uncompress it into show file.xml and source file.dex. At that point on passing the source file.dex to dis-assembler it will remove the standard Programming interface utilized as a part of the application records. Other than Programming interface, figure 4.5 work call additionally assumes an essential part in deciding the pernicious exercises. Get together can be better than Programming interface brings in that it permits a more definite correlation of executables. Programming interface calls, then again, can be better than Gathering for its speed and its littler mark.

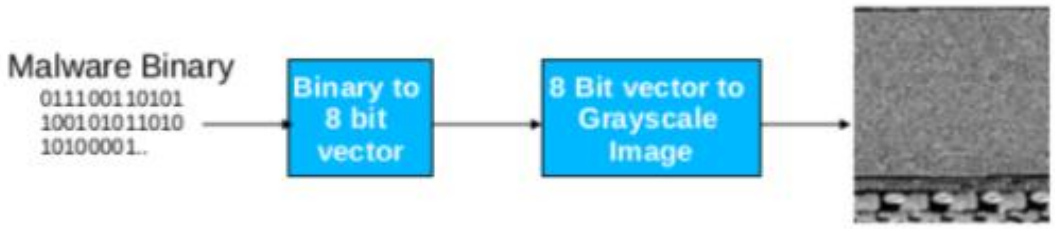


Figure 4.5 Resolving API calls via executable image

4.6 Malware as an image

This is a quite unique way to represent the malware executable as a grayscale image for purpose of detection and classification. In this method malware executable represented as a binary string consisting of zeros and ones (0,1) .Then this vector of zeros and ones turned into a matrix of specified dimension , this matrix behave as a grayscale image used for classification and detection of malware present in the file . To classify the malware you can use K-nearest neighbor algorithm with Euclidean distance and to compute the texture features we can use GIST.

4.7 Use of registers

This method uses the values stored in registers when a malware program is running. So the values obtained during the running of malicious and benign software can be work as a distinguishing feature. On each API call we can store the values of registers before and after the API calls. After that trace the changes occurred in register values before and after application programming interface is called. The make a vector obtained from the values of registers like EAX, EBX, EDX, EDI, ESI and EBP registers. Final results demonstrated by comparing the old and new vector for unseen malware.

4.8 Call Graphs

It is a direct graph representation of a program in which it shows that how the subroutines calling each other. Here each node represents the subroutine and edge represents that who is calling whom. So here we have to design a framework to build function call graph from the information extracted through disassembled malware file. For each hub (i.e. function) in the graph, they removed properties including library APIs calls and what number of I/O read operations have been made by the work. At that point, they figured the likeness between any two malware product cases.

Chapter 5. Results & Evaluation

We have utilized the distinctive classification strategies and utilize the Android application Permissions and APIs as the elements for order display. We have gathered the 440 test of the malware and the generous application, utilized the Androguard [56] and Mosf [40] to extricate the consent and Programming interface from the android application bundle in particular .apk document and made a dataset for the characterization model, for example, Decision tree, Neural Systems.

5.1 Implementation

The android application bundle .apk record are handled with Androguard to remove the highlights Android authorization and APIs and gathered the consent to make a dataset what's more, aggregate have the 330 elements. We have utilized the Quick Excavator apparatuses [57], the python Scikit-learn [27] and Perfect python to play out the order on the element dataset of the authorizations and APIs. Presently we will talk about outcomes and assessment.

5.1.1 Results

We had connected the Decision tree, Neural Systems characterization strategies, we had utilized the dataset with respect to the preparation and testing is finished 10 cross overlap. We talk about the aftereffect of every method one by one and they analyze them with the Flawless neural systems.

Decision Tree

5.1.1.1 Using Gini Index

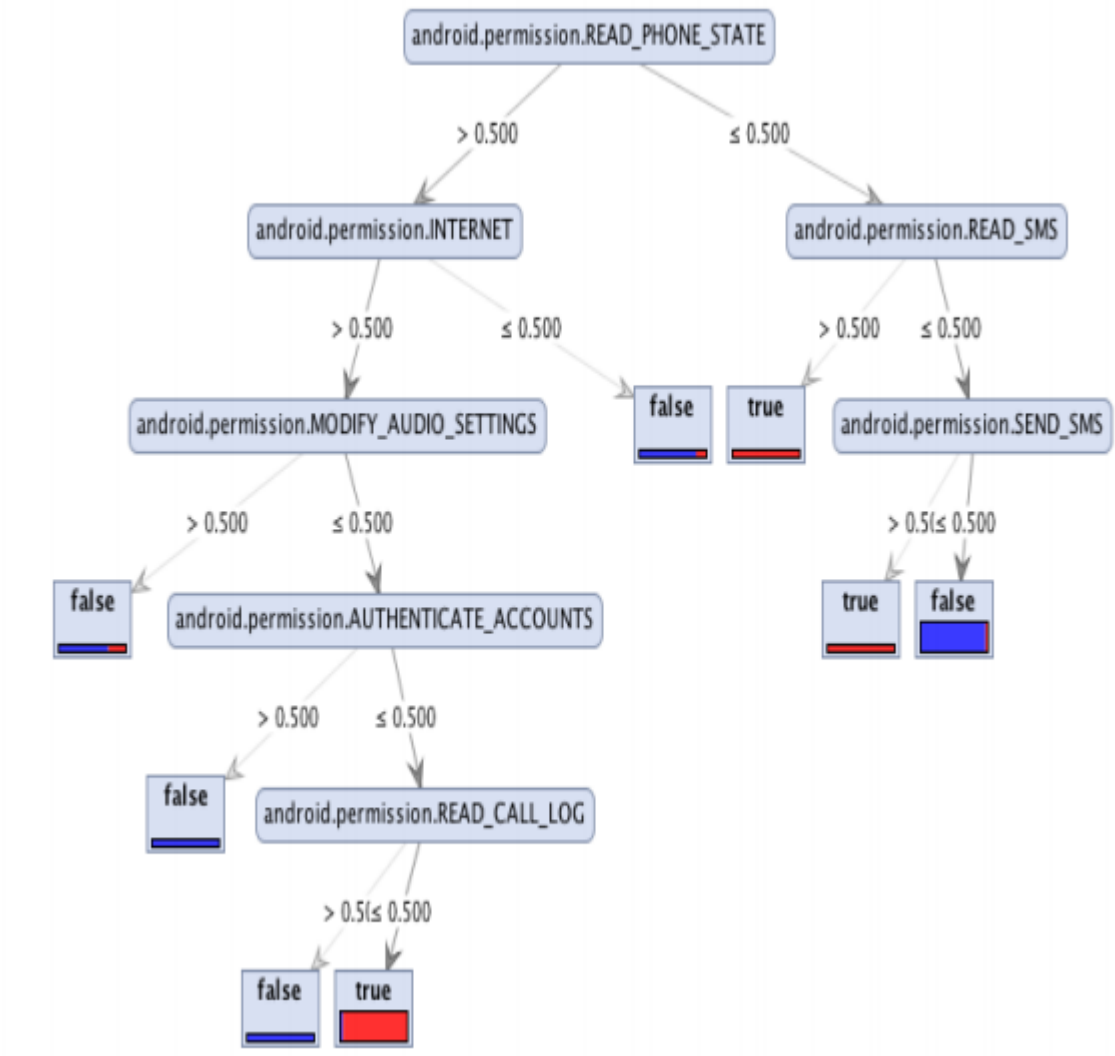


Figure 5.1 Decision Tree with Gini index

Item NO.	Accuracy	Precision	Recall	F1_Score
1	74.4	69.9917557932	74.4	70.3844038718
2	74.4	61.3957375479	74.4	64.5601616162
3	62.0	62.1556709957	62.0	57.0238222971
4	77.5	78.8874895572	77.5	77.1946742532
5	71.3	66.3924006075	71.3	64.3761920373
6	83.7	82.068226817	83.7	75.769352166
7	62.0	60.1655723906	62.0	55.311356503

Table 5-1 Results Using Decision tree with Gini Index

5.1.1.2 Using Entropy

Item NO.	Accuracy	Precision	Recall	F1_score
1	74.4	72.9134528515	74.4	70.1922524834
2	89.9	76.1635555556	89.9	80.4600222173
3	68.2	67.99553467	68.2	64.0776219244
4	80.6	72.2709687681	80.6	74.9314549791
5	71.3	67.0062394958	71.3	66.2554926547
6	80.6	83.1998354626	80.6	69.9507987193
7	65.1	57.2436100132	65.1	57.8311038752

Table 5-2 Results Using Decision tree with Entropy Calculation

Decision tree with Entropy example:

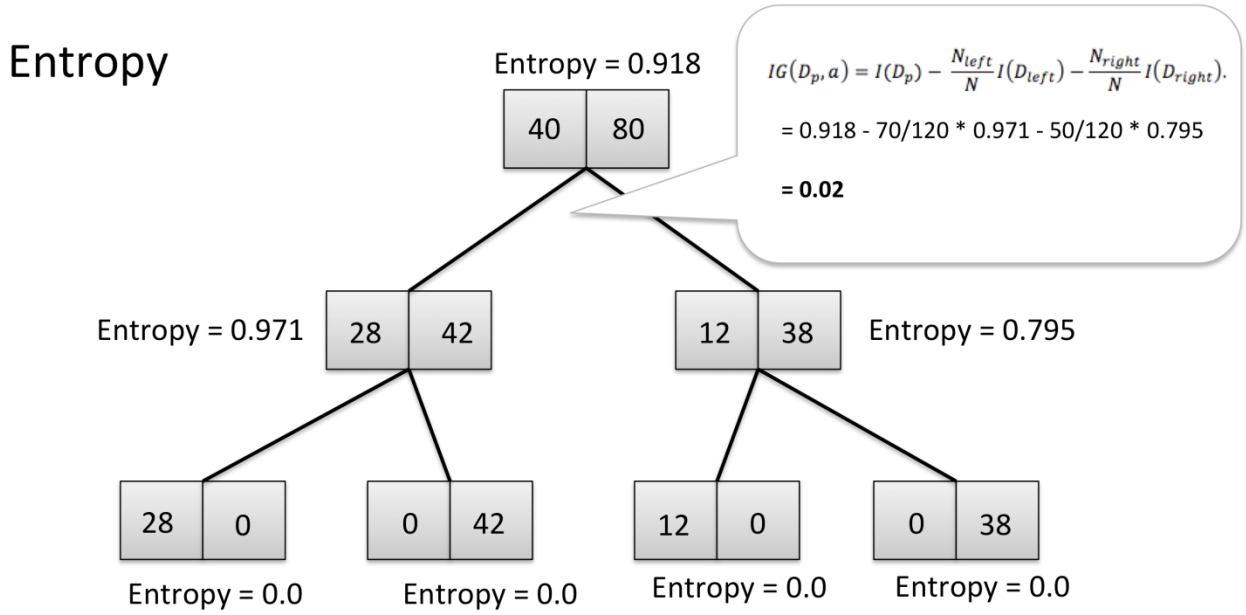


Figure 5.2 Entropy Calculation for Decision tree

Support Vector Machine

5.1.1.3 Using Linear SVM

Item No.	Accuracy	Precision	Recall	F1_score
1	71.3	72.1147005772	71.3	70.472194211
2	74.4	82.2106165171	74.4	70.2916233766
3	71.3	55.9106636156	51.3	62.024477738
4	65.1	60.4371505687	65.1	61.1400462154
5	65.1	54.4060121038	65.1	58.0587163268
6	77.5	66.7503839467	77.5	68.7402424286
7	62.0	63.2480519481	62.0	59.7253316503

Table 5-3 Results using Linear support Vector machine

5.1.1.4 Using Gaussian Kernel

Item No.	Accuracy	Precision	Recall	F1_score
1	71.3	69.3375831486	71.3	65.1864242424
2	55.8	55.0208189446	55.8	51.9783888889
3	68.2	56.4942428978	68.2	58.84746468
4	52.7	61.8216993464	52.7	55.7309845074
5	74.4	73.8567401961	74.4	71.7055018234
6	80.6	76.1844537815	80.6	76.1439451115
7	58.8	49.0083093377	58.8	52.8533972396

Table 5-4 Results using Gaussian Kernel

5.1.1.5 Using Sigmoid Kernel

Item No.	Accuracy	Precision	Recall	F1_score
1	43.4	6.076	43.4	10.6596491228
2	37.2	4.464	37.2	7.97142857143
3	46.5	6.975	46.5	12.1304347826
4	55.8	40.9672779923	55.8	40.3244586201
5	55.8	21.3577922078	55.8	30.7632775633
6	49.6	7.936	49.6	13.6827586207

Table 5-5 Results using sigmoid Kernel

Neural Systems

5.1.1.6 Using Neural Networks

Item No.	Accuracy	Precision	Recall	F1_score
1	88.0	101.523809524	88.0	89.6727272727
2	56..0	50.4761904762	56.0	50.819047619
3	72.0	93.9487179487	72.0	76.2666666667
4	32.0	45.6857142857	32.0	35.0099865047
5	72.0	61.3333333333	72.0	57.8517482517
6	72.0	85.3333333333	72.0	75.7333333333
7	96.0	99.3841269841	96.0	95.9327731092

Table 5-6 Results Using neural Network

5.2 Evaluation

We have thought about the order consequences of the different methods with NNs, We Have measured Accuracy, Precision, Recall and F1-score of the every characterization with the neural networks. Table 5.8 demonstrate the normal score of the every systems we have connected in the arrangement of investigation performed and found that the NNs had the most extreme of 80.142 % of exactness from the various one is best from rest of methods.

The ROC bend was utilized appeared in figure 5.7 to quantify the score for the diverse methods, we find that the Support Vector Machine was having the bend near the NNs

remarkable from the Decision tree and Neural Network, however it was not capable to get a similar closeness to the 1 like as of the NNs.

We have additionally plotted the diagram to gauge the exactness rate for each of the procedures also, find that the NNs had the exactness bend which is reliable and is all the more every now and again touching the most extreme estimation of 100% in figure 5.8, in this manner have the greatest likelihood of getting the expectation with most extreme precision.

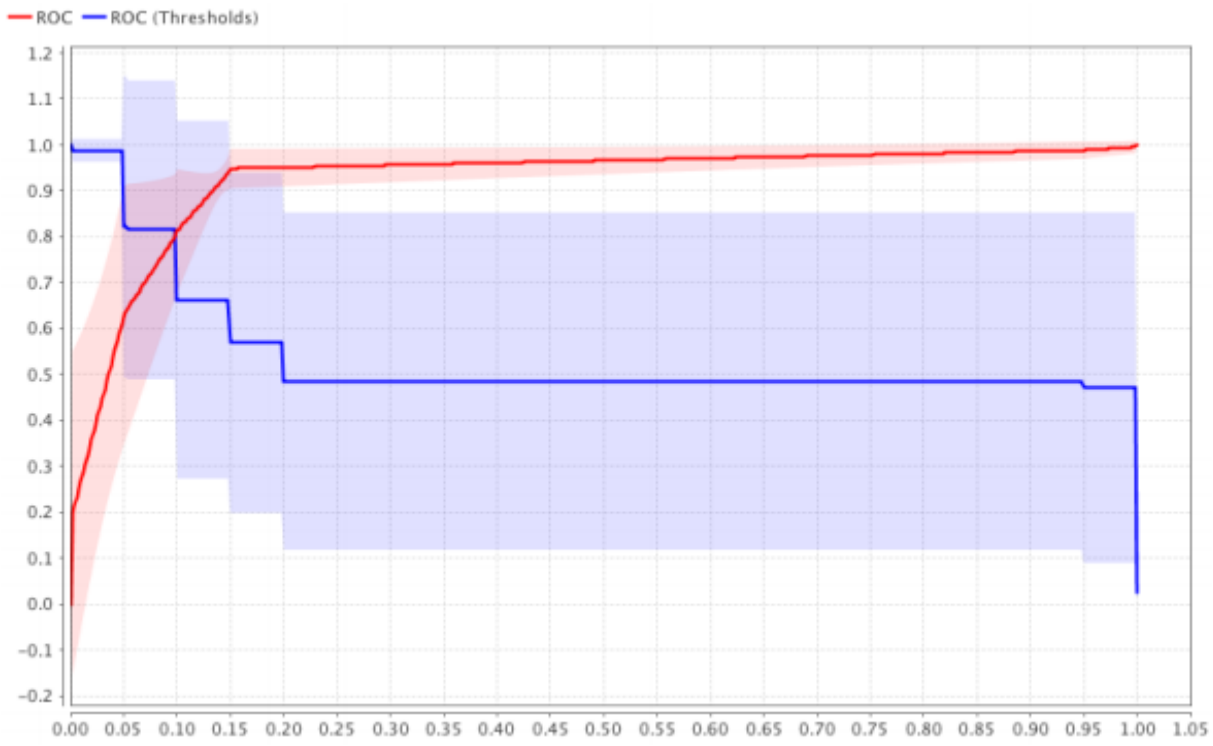


Figure 5.3 ROC for decision Tree

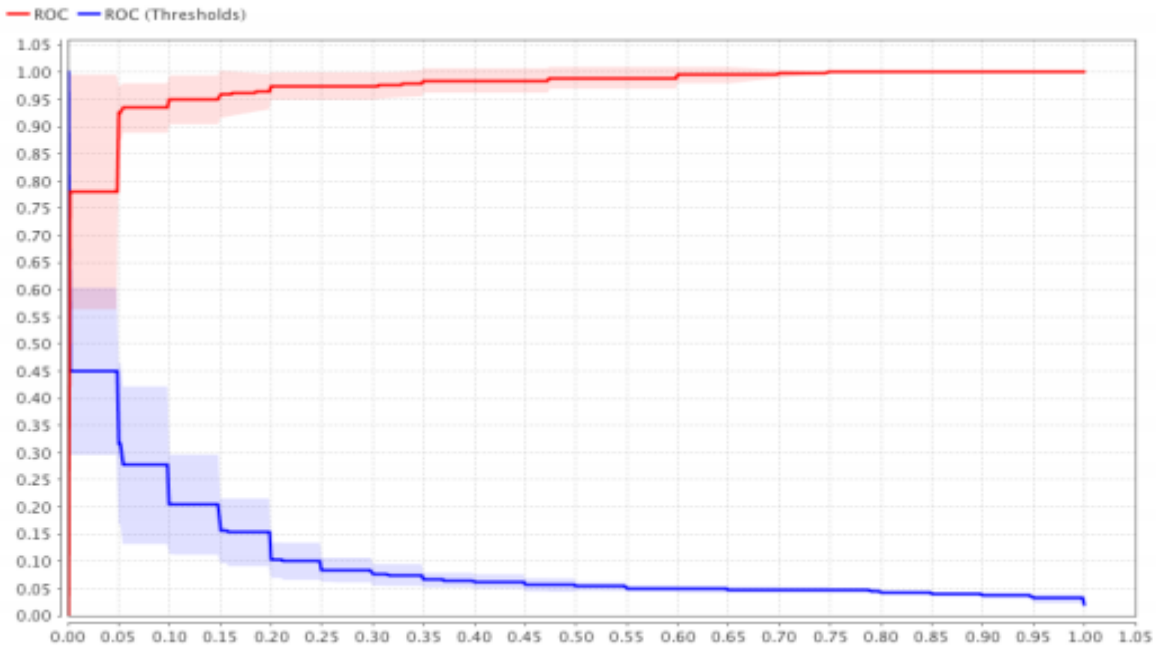


Figure 5.4 ROC for SVM

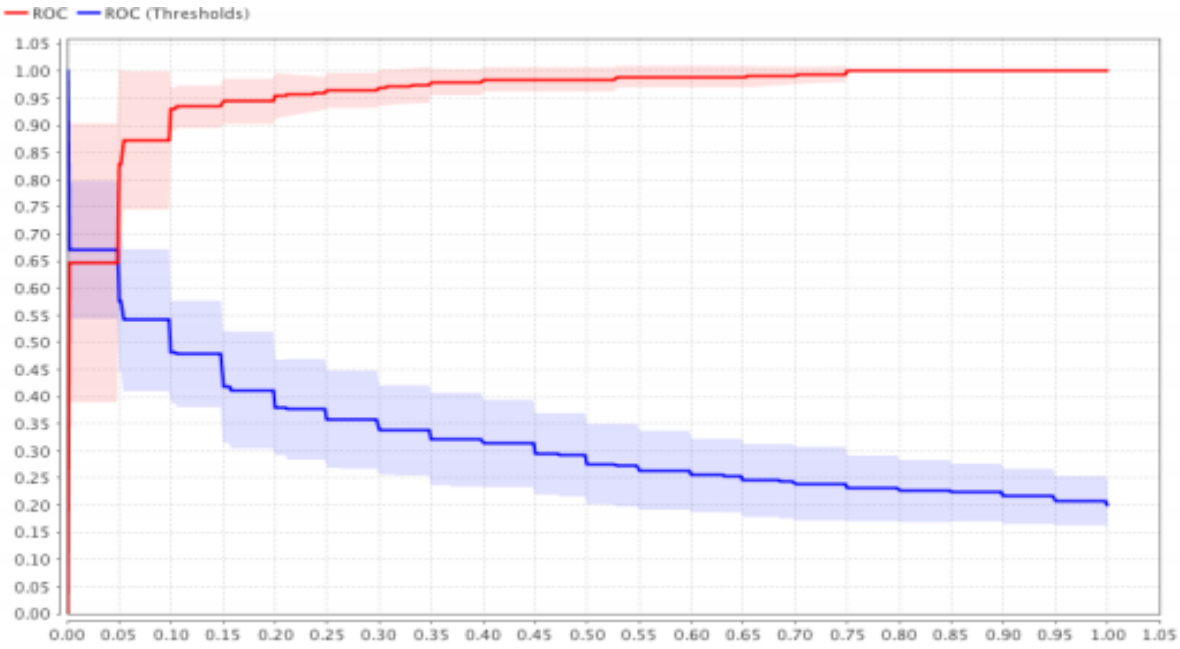


Figure 5.5 ROC for Neural Networks

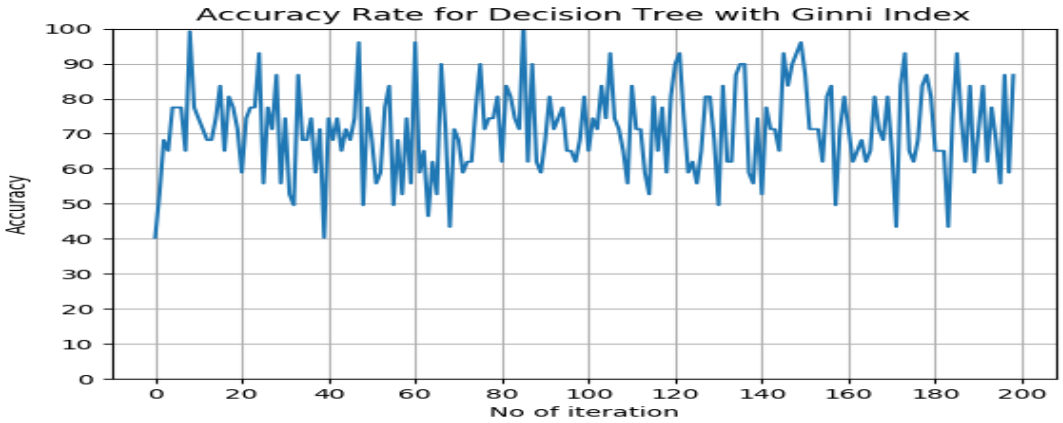


Figure 5.6 Accuracy rate for Gini index based Decision Tree

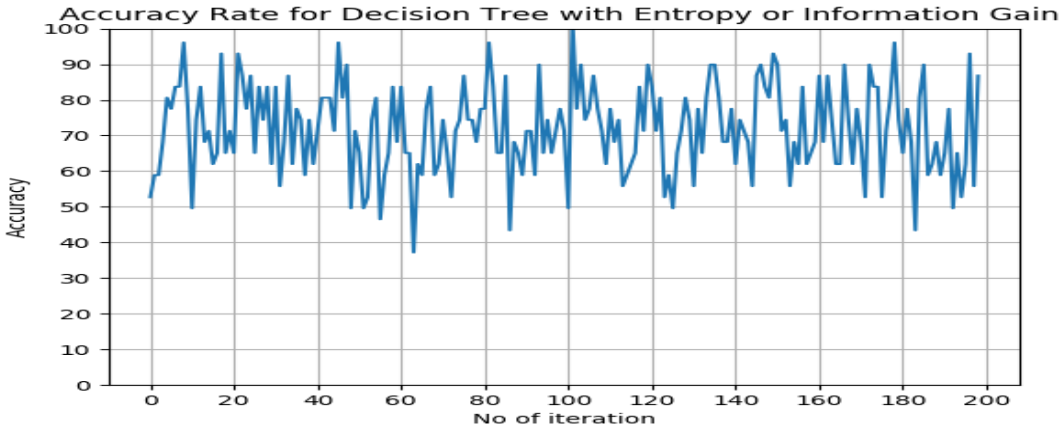
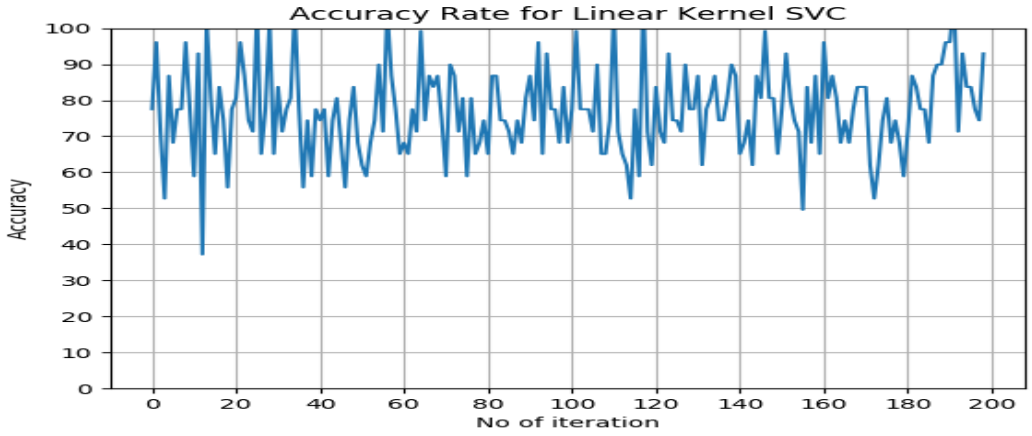


Figure 5.7 Accuracy rate for information Gain based Decision Tree



.

Figure 5.8 Accuracy rate for Linear Kernel based SVM

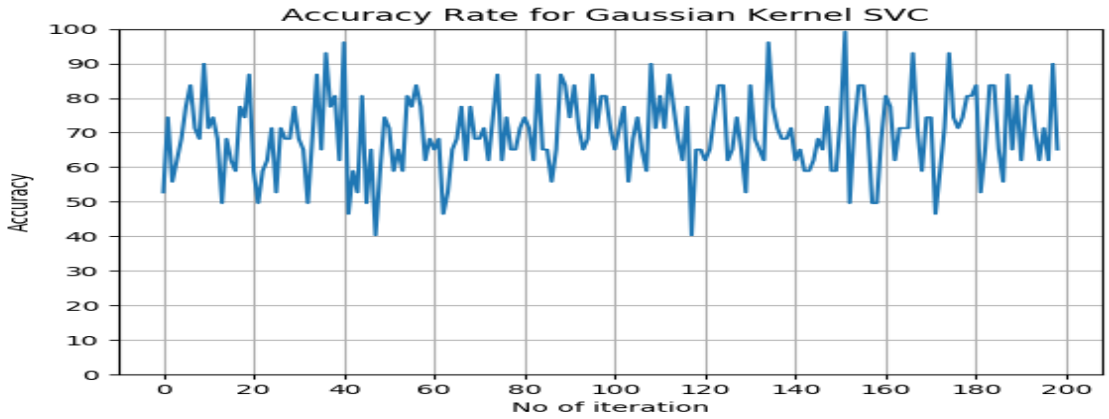


Figure 5.9 Accuracy rate for Gaussian Kernel based SVM

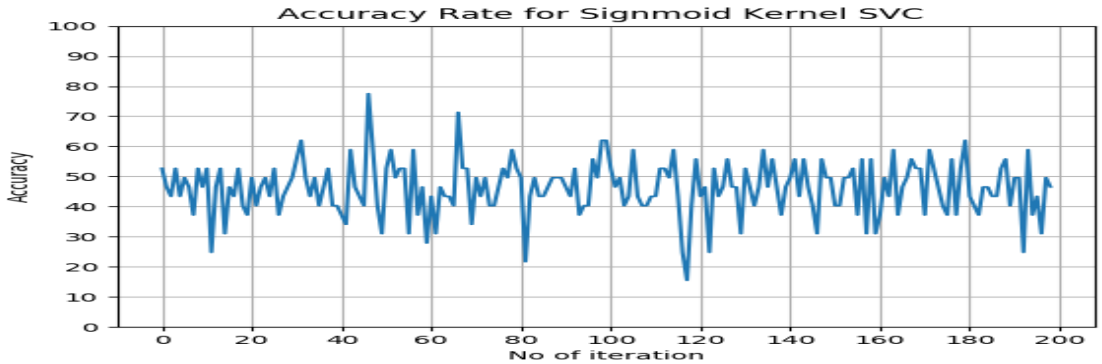


Figure 5.10 Accuracy rate for sigmoid kernel based SVM

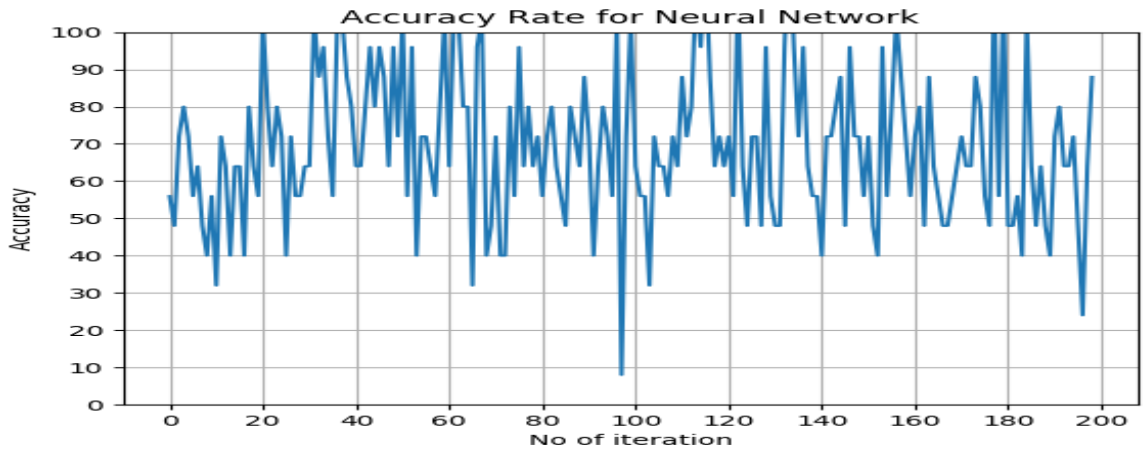


Figure 5.11 Accuracy Rate of Neural Networks

Summarization:

So based on the provided results, on the predefined dataset neural networks works better and we achieve 81% success by using it. Whereas other techniques like decision tree and support vector machine produces results nearly 60% and 75% simultaneously. Neural networks outperform other algorithms like logistic and linear regression, but in order to increase its efficiency more we have to apply more number of hidden layers. Such that model presented by this neural networks works better. Figure 5.11 shows the neural networks behavior, based on different data values the average prediction comes around 80% which is much better than other techniques. If we increase the number of hidden layers in neural nets than prediction accuracy reached 96% where this model behaves like deep neural networks. Accuracy rate of neural system also depend on the no. of iterations sometimes prediction rate becomes better on increasing the no. of iterations whereas sometimes it decrease due to increase in no. of iterations .

As SVM is used for binary classification, so based on linear, Gaussian kernel and sigmoid SVM we can measure the prediction rate of different techniques. Figure 5.8 shows that linear SVM

works better on binary classification in order to detect a malign or benign software. Gaussian kernel shows prediction efficiency average 75%, which shows that even on increasing the no. of iterations prediction rate still same Figure 5.9 demonstrate that. Figure 5.10 shows that sigmoid kernel have prediction rate even lesser than linear and Gaussian kernel. Sigmoid kernel has 45% on average prediction accuracy rate.

Figure 5.6 shows the prediction rate using Gini index which shows average 60% prediction rate and here it basically depends on attribute selection on starting of operation. Figure 5.7 shows the decision tree prediction rate with respect to entropy or information gain which accounts nearly 70-75% prediction rate, much better than Gini index and sigmoid kernel.

Chapter 6. Conclusion & Future Work

6.1 Conclusion

Malware of any stage either on normal desktop PC or portable stage like android, iOS and so forth., are exceptionally hard to distinguish in this present reality situation, the standard signature based systems are not that much viable and end up plainly pointless when the new zero day assault comes in. Despite the fact that the dynamic investigation are superior to the mark based one yet they requires the cloud benefit or the other virtual condition to execute the malware , yet there is meagerness of administration which could do computerized dynamic investigation of malware. Likewise the malware can distinguish the virtual condition and change their conduct. In this work we propose a pattern for the location of android malware by utilizing the API and Permission of the android applications and utilized the far more propel system for machine learning than the standard procedures called NEAT, which is a kind of support realizing which has a high detection (classification) rate furthermore, create an advanced structure which give the speedier recognition because of quicker preparing.Examination on this present reality show the great execution of the Slick concerning the neural systems, Support vector machines and Decision trees utilizing the android consent and API as the elements.

6.2 Future Work

In future we will jump at the chance to broaden our work by including and finding the new component from the use of the android also can utilize more propel machine learning methods like deep learning method to increase the number of neurons in

hidden layer . For the components we can utilize the hex dump picture of the apk, bump record in the apk, database record if introduce in the apk, handle dump of the application in different time interim amid execution, and afterward speak to them as the picture of N x N lattice. We can utilize this picture as the element for the machine learning. For the machine learning system we can utilize the Deep Learning method by applying the convolution arranges on those pictures what's more, other preparing and play out the preparation, with this systems we can utilize the computational energy of Graphic Processing Unit alongside Central Processing Unit to increment the rate of learning, and afterward we can port this model to the android gadget which can recognize the malware.

Chapter 7. Bibliography

[1] Wikipedia, “Google Play — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/wiki/Google_Play, 2017.

[2] Androguard 2017 https://www.darknet.org.uk/2016/11/androguard-reverse-engineering-malware-analysis-for-android/ .

[3] Android services https://developer.android.com/guide/components/services.html .

[4] Wikipedia, Android application package - https://developer.android.com/reference/android/app/package-summary.html.

[5] APK Tool- https://github.com/iBotPeaches/Apktool 2017.

[6] Android dataset permissions –Kaggle https://www.kaggle.com/xwolf12/datasetandroidpermissions 2017.

[7] IEEE malign or benign dataset permission android – IEEE Journal https://ieee-dataport.org/documents/dataset-malwarebeningn-permissions-android 2017.

[8] Yearly analysis for the sale of android and windows sales - http://www.zdnet.com/article/canalys-predicts-a-billion-android-smartphone-sales-in-2017-and-rapid-growth-for-windows-phone/ 2017.

[9] Usage share of android operating systems – Wikipedia https://en.wikipedia.org/wiki/Usage_share_of_operating_systems 2017.

[10] Security report over malware by alcatel leucent -https://resources.alcatel-lucent.com/theStore/files/Kindsight_Security_Labs_Q112_Malware_Report_EN.pdf 2017.

[11] Android developer security best training - https://developer.android.com/training/articles/security-tips.html 2017.

[12] Application verification service in android –Xiang University https://www.csc2.ncsu.edu/faculty/xjiang4/appverify/ 2017.

[13] file system footprints used in malware classification http://ieeexplore.ieee.org/document/7501294/ 2017.

[14] Smart phones malware security case – IEEE Journal http://ieeexplore.ieee.org/document/6732369/ 2017.

[15] TWMAN –behavioral analysis of malware IEEE journal http://ieeexplore.ieee.org/document/5953604/ 2017.

[16] characterization and evolution dissecting android malware - http://ieeexplore.ieee.org/document/6234407/ 2017.

[17] Android malware measures and counterattacks : current and future directions http://ieeexplore.ieee.org/document/6992944/ 2016 .

[18] Network Traffic Analysis used for malware analysis for mobile devices :http://ieeexplore.ieee.org/document/6982893/ 2016.

[19] Security Breach and android applications –IEEE journal paper http://ieeexplore.ieee.org/document/7529383/ 2015.

[20] Research on android malware detection based on network traffic monitoring - http://ieeexplore.ieee.org/document/6931449/ 2013.

[21] Malware detection evolution technology A study approach :http://ieeexplore.ieee.org/document/7389671/ 2011.

[22] Android Manifest file based android malware classification approach :http://ieeexplore.ieee.org/document/7790261/ 2010.

[23] Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O :Online machine learning algorithm based android malware classification approach - https://link.springer.com/chapter/10.1007/978-3-319-47217-1_8 2008.