Bachelor of Science in Computer Science & Engineering

Prediction of Android Malware Families Utilizing Smartphone Data

by

Pranto Kumar Biswas

ID: 1404076

Department of Computer Science & Engineering

Chittagong University of Engineering & Technology (CUET)

Chattogram-4349, Bangladesh.

April, 2021

Prediction of Android Malware Families Utilizing Smartphone Data

Submitted in partial fulfilment of the requirements for
Degree of Bachelor of Science
in Computer Science & Engineering

by

Pranto Kumar Biswas
ID: 1404076

Supervised by

Dr. Md. Iqbal Hasan Sarker
Assistant Professor
Department of Computer Science & Engineering

Chittagong University of Engineering & Technology (CUET)
Chattogram-4349, Bangladesh.

The thesis titled ' **Prediction of Android Malware Families Utilizing Smartphone Data'** submitted by ID: 1404076, Session 2019-2020 has been accepted as satisfactory in fulfilment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering to be awarded by the Chittagong University of Engineering & Technology (CUET).

Board of Examiners

Chairman

Dr. Md. Iqbal Hasan Sarker
Assistant Professor
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Member (Ex-Officio)

Dr. Asaduzzaman
Professor & Head
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Member (External)

Muhammad Kamal Hossen
Assistant Professor
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Declaration of Originality

This is to certify that I am the sole author of this thesis and that neither any part of this thesis nor the whole of the thesis has been submitted for a degree to any other institution.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. I am also aware that if any infringement of anyone's copyright is found, whether intentional or otherwise, I may be subject to legal and disciplinary action determined by Dept. of CSE, CUET.

I hereby assign every rights in the copyright of this thesis work to Dept. of CSE, CUET, who shall be the owner of the copyright of this work and any reproduction or use in any form or by any means whatsoever is prohibited without the consent of Dept. of CSE, CUET.

Pranto Kumar Biswas

Signature of the candidate

Date:18/4/2021

Acknowledgements

First and foremost, I would like to thank God for the good health and well-being that were necessary to complete this work. I would like to express my sincere gratitude to my honorable project supervisor **Dr. Md. Iqbal Hasan Sarker** *Assistant Professor, Department of Computer Science & Engineering, Chittagong University of Engineering & Technology* for providing me the perfect guidance, encouragement, instructive suggestions with all the necessary facilities for the research and preparation for the thesis. I place on record, my sincere thank you to **Dr. Asaduzzaman,** *Head and Professor of the Department, Department of Computer Science & Engineering, Chittagong University of Engineering & Technology*, for the kind encouragement. I am also grateful to our external, **Muhammad Kamal Hossen** *Assistant Professor, Department of Computer Science & Engineering, Chittagong University of Engineering & Technology*, for his guidance and review of our project. I take this opportunity to express gratitude to all of my Department faculty members and seniors for their help and support. I also thank my parents for the unceasing encouragement, support, and attention. I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Abstract

Being the most popular smartphone operating system, Android provokes the attackers to create advanced malware threats. So to provide appropriate defense against malware attacks, malware must not only be detected but also be categorized into families analyzing their functionality. Any program when it is executed on a device starts creating its characteristic footprint on hardware. Therefore hardware activity is a great source of features. The main goal of our work is to classify the android malware application samples into their families. For this purpose, we propose to use an effective set of ram usage features. Using these features a dataset is processed from the memory usage pattern of real phone android applications. According to experimental results, we show that the extracted features contain enough information to detect the malware families with an accuracy of 72%. As our approach relies on the fewer number of features than API and network flow based dataset for the machine learning model, it provides a faster and less complex malware family detection technique.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Android malware in the form of malicious apps are developed with the intention of stealing data, gaining access or causing damage to users' smartphone devices. Day by day the attacking technique are changing with such a rapid manner that the upgrade versions of Android OS are still facing strong malware threats. Google Play Store[1], the primary source for installing android apps is the most secure online platform, where the apps' source code are checked in frequently whether they contains any vulnerable lines or not. But attackers use such advanced evasion techniques that they doesn't need to depend only to coding smartly.

The safety measures considering static behaviors i.e permissions, intents, certifications, source code has been analysed over and over. Types of malware include adware, worms, Trojans, spyware, ransomware behave in a way that the dynamic detection technique comes into play. Dynamic code loading causes to add payload viruses in run time[2]. To tackle this type of problems an update dataset with recent attack is needed along with faster detection technique.

To achieve the best result in detecting and then blocking the malware apps, the families of the malware are also needed to be detected [3]. Because same family member shows same characteristics. At that case, classifying the similar family member makes the post attack technique against the malware more faster.

Having different dynamic behaviors (network flow, API, sys. call, battery log, memory log, CPU usage), analysing hardware level data is mostly impossible to invade for malware. Because the app's activity keeps the footprint in the hardware all time. Even dynamic code loading can't obfuscate the hardware

based detection technique. So we take into consideration the real smartphone based hardware data for detecting malware's family. Our work focus on utilizing the low level data to select a optimal set of features so that the model can predict the vulnerability of an app more quickly.

1.2 Applications

Malware detection and their family classification is the duty of an anti-malware software. They protects the smartphone users' devices the from being hacked.

Both online and offline, the users' get virtual threats from the malicious application. So the malware detection in background process helps him to use the devices without becoming worried.

Data is the most precious thing in this world. Malware detection system gives them the security by protecting them from being stolen. The faster the detection technique, the faster the prevention starts up, less data is leaked. That's why anti malware software depends on optimal feature set based detection technique.

1.3 Motivation

Anti-malware systems need faster technique to efficiently identify the malware. If the malware are grouped into their families, then the anti-malware system does not need to do individual post attack for every single app. A batch post technique can be applied then. In order to grouping we need family classification. Moreover accuracy plays a great role here also. Having large feature set make the detection system slower.

As far as we know, there is no ram usage dataset which is based on hardware level data and collected from practically installed on smartphone. In 2019 a research work captured the ram usage data from smartphone[4], but is not processed for prediction. As the hardware level data keeps track an app's work flow, they also keep the malware's behavior footprint. It inspires us to analyses the ram usage data and to provide a minimized feature set for family classification.

1.4 Difficulties

The behavior of malware are updating day by day. Their detection nowadays become difficult because advanced obfuscation technique. The main difficulty to detect the malware is to process a dataset from the captured text data of memory usage pattern of the real smartphone.

It is difficult to interpret the hardware level data in case for ram usage. The different part of ram i.e physical memory needs to study a lot to understand the significance of the features.

1.5 Contribution of the thesis

The primary focus of our work is to predict the families of malware using supervised machine learning model. The contributions are as follows,

1. We processed a ram usage features based dataset for malware apps having optimal set of features by processing semi structured data.
2. We provide a correlation based feature extraction technique for multi-class data to predict android malware families .

1.6 Thesis Organization

This report is organized into five chapters. Chapter two contains a brief discussion on previous works that are already implemented, their limitations, and their role in malware classification using machine learning. Chapter three describes the proposed method with necessary diagrams. An overall system architecture is given in this chapter. Also, our implementation of the project in detail has been illustrated. Chapter four focuses on the experimental results of the technique. Evaluation measures and results of our system are described in this chapter. Chapter five consists of a conclusion with a summary of our system and the future plan for our proposed method.

Chapter 2

Literature Review

2.1 Introduction

Research on suspicious software detection is carried on by several significant researchers. In this chapter we describe the previous research on android malware detection and their present state.

2.2 Related Literature Review

The genome project[5] is among the very earlier publicly available android malware dataset which is outdated after 2015. Though the dataset contains 1260 malware samples, the features are based on permission and source code based static behavior. To overcome the scarcity of the dataset in 2017, the AMD dataset [6] was introduced having 405 samples of 71 malware families which followed the static behaviors also.

S. Turker et al.[7] classified the families based on the features-API and permissions of AMD dataset [6]. Where the authors came to find some unknown malware family detection considering the confidence value. The SVM has done best for them.

Researchers of [8] worked on the famous Drebin dataset having 179 families. But they processed the Drebin dataset on emulator environment. To predict from this very large scale dataset they proposed the EC2 algorithm whose performance is up to the mark. But emulator based detection can't detect many real life malware. Because when an app can identify that it is running on an emulator then it starts to hide its malicious behavior.

Since update attacks load their malicious code at runtime, K. Aktas et al.[9] tried to fusing both static and dynamic features and introducing another emulated dataset named UpDroid. Where they got 97% accuracy. Their performance matches poorly with the real attack. But There research contains only 20 family.

To extract the features from the running app a novel framework AndroPyTool is made by A. Martín et al.[10] . That paper showed that weighted average on extracted features can achieve 89% accuracy on malware family classification. Again this is based on emulation like dendroid[11] and droidsieve[12].

To overcome the emulation, in [13], N. Kiss et al. worked on the real phone based data which has only seven malware family, where there is no malware family detection method. They just provide the dataset to the researchers for analysing and predicting malware.

Another real smartphone based dataset provided by Lashkari et al. [14] in 2017 named AAGM. This dataset contains 9 network traffic features of 400 malware samples with 1500 benign sample. But their classification is limited to binary classification and only categorised the adware types. The work [15] focused on developing a standard dataset for malware detection and classification. More than 5000 thousand application were installed and the app's various static and dynamic behavior were captured. Later on, following the previous work Taheri et al.[4] provided a completed dataset for malware detection and family classification which contains API, network flow and permission features. Other behaviors i.e ram usage, battery usage and CPU usage were kept in raw form. But their dataset contains a huge feature set about 8000 in number. They have done three types of classification, binary, malware category and malware family. But the family classification accuracy is about 60%.

Some work have proposed to use hardware level data to detect malware. Among them Massarelli et al. [3] proposed to use resource consumption metrics from the PROC file system. But again it used the emulated drebin dataset. In [16] Banin et al. shows that low level features such as memory access patterns can detect malware with higher accuracy though they focused on desktop version. The researchers of [17] for the first time used the ram usage data using the

Table 2.1: Summary of captured features of some available android malware dataset

Dataset Name	Published Year	**Number of benign samples**	**Number of malware samples**	Permission	Source Code	API. Call	Network	Sys. Call	Log	Ram Usage	Installed on
Andro-dumpsys	2016	1176	906	yes	yes	yes	no	no	no	no	Emulator
Kharon	2016	-	7	no	no	no	yes	yes	yes	no	Real Phone
AAGM	2017	1500	400	no	no	no	yes	no	no	no	Real Phone
AMD	2017	-	405	no	Yes	no	no	no	no	no	Emulator
CICAndMal2017	2018	1700	426	yes	no	yes	yes	no	no	no	Real Phone
CICAndMal2019	2019	1700	426	yes	no	yes	yes	no	yes	yes	Real Phone

outdated genome [5] dataset. They provided the important features for the binary classification. In the Table 2.1 the publicly available dataset's are included with their contained features.

From the above mentioned works, it is clear that the malware family classification is done in a wide range. But none of them focus on the use of ram usage behavior of real smartphones. Through our proposed work we classify the malware family by processing a dataset use of captured ram usage data captured in [4] properly.

Chapter 3

Methodology

3.1 Introduction

In this chapter, we discuss our proposed methodology, implementation and learn about every module of our system. Detailed mathematical terms are described also.

3.2 Proposed Methodology

The objective of our work is to classify the families of Android malware apps. For this, Our proposed methodology has two major parts. One is data preprocessing and another one is model building. Figure 3.1 shows the outline of our work.

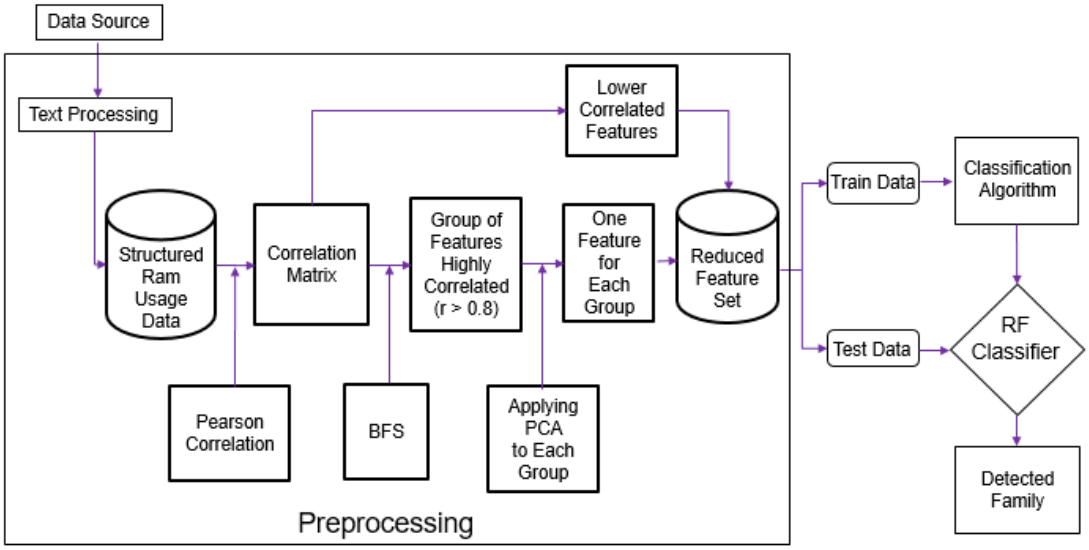


Figure 3.1: Proposed model to classify malware family

3.2.1 Preprocessing

In machine learning, data preprocessing is a crucial stage. Data preprocessing is a data mining technique that entails converting raw data into a format that can be understood. Real-world data is not often complete, unreliable, and deficient in specific has patterns, as well as containing numerous errors. Preprocessing data is a trusted way of addressing such problems. To fill in values that are missing, filter noisy data, recognize or eliminate outliers, and fix inconsistencies, data preprocessing is needed.

Our source of data contains many files having text data. So, words that have no meaning should be omitted from the text files. The remaining text contains data that are going to be included in the dataset. For text data processing regular expression searching pattern is used in our work.

3.2.1.1 Text data processing from raw data

A supervised machine learning model needs well-labeled structured data to be trained. The collected raw data is in form of text which is semi-structured. To get insight from the text data it is recommended to convert the text data in a structured form. We have used regular expression technique to split the strings from the text data. It helps us to get the feature names and the value for the features.

3.2.1.2 Regular Expression

A regular expression is a string of characters that defines a pattern to be searched for. It is often known as regex. Each character in an expression (each character in the pattern of the string described) is either a metacharacter having a special meaning or a regular character with a literal meaning. For example, rub[ye] will match for “ruby” or “rube”, \s matches for a whitespace character, \S matches for nonwhitespace character, [0-9] matches any digit same as [0123456789].

In python, there are two primitive operations for regular expressions: ‘Match’ checks for a match only at the beginning of the string, while ‘Search’ checks for a match anywhere in the string.

The match function matches RE *pattern* to *string* with optional flags.The search function searches for first occurrence of RE *pattern* in *string* with optional flags.

The regular expression is used in our work to generate the name of the features and the feature's numerical values for the samples.

3.2.1.3 Feature extraction

Feature extraction is a fundamental concept in machine learning that has a significant effect on the model's efficiency. In the real world, it's unlikely that all of the features in a dataset are useful for constructing a machine learning model. Adding redundant features decreases the model's generalization potential and can also decrease a classifier's overall accuracy. Furthermore, adding more features to a model increases the model's overall complexity. For our approach, we firstly select some features to be extracted and the other features are kept as original.

Feature selection: Feature selection is the study of algorithms to reducing the dimensions by selecting a subset of relevant features to improve machine learning performance. For a dataset with N features (or attributes), feature selection targets to reduce N to N$'$ and N$'$ $\leq$ N.

After getting a feature set applying regular expression, the dataset contains so many missing values, redundant data. The features having missing values, constant values more than 95% are dropped.

- **Filter Method:** Filter feature selection methods give a score to each feature using a statistical calculation. The features are graded and either retained or removed from the dataset based on their ranking. The methods are frequently univariate, taking into account most the independent feature.

  The Chi-squared measure, correlation coefficient scores, and information gain are examples of filter methods.

Considering the computational cost we have used the filter method. After getting the feature set removing the missing data we move forward to select an optimal set of features from *Structured Ram Usage Data*. From the 3.1 diagram the next

step is to make a correlation matrix. To create a correlation matrix we use the Pearson Correlation method.

- **Pearson Correlation:** Pearson correlation coefficient or Pearson's correlation coefficient or Pearson's r is defined in statistics as the measurement of the strength of the relationship between two variables and their association with each other [18]. In simple words, Pearson's correlation coefficient calculates the effect of change in one variable when the other variable changes.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

Applying the correlation formula we get a correlation matrix. In the matrix, the highly correlated features show the r-value near about 1 or -1. It is assumed that a high correlation is above 0.9 or below -0.9. To find which features are highly correlated with others, we apply the breadth-first search algorithm.

- **Breadth First Search Algorithm:** Breadth first search or BFS algorithm is an efficient graph-based searching algorithm. This algorithm search level by level.

  In BFS, traversing starts from a selected source or starting node and traverses the graph level wise to explore the neighbor nodes which are connected to the source node. For our purpose, we use the BFS algorithm modified in a bit. The traverse happens for each value of the matrix. And we use the adjacency matrix for the edge. So the worst case time complexity of the algorithm is $\mathcal{O}(n^2)$.

  The motive to use the BFS is to find out some groups. In each group, the group members are highly correlated to each other.

  The traverse starts from the first element(source node) of the correlation matrix and for each feature connected with the first element are checked whether their r-value is above 0.9 or below -0.9 or not. Those who maintain the condition are added in a queue as second-level nodes for the first

element. In such a way the BFS finds highly correlated features in a group for source node feature.

Those features which are not visited yet are considered for source node one after one and grouped features correspondingly following the above traversing method.

Identifying the groups, from each group one feature is extracted. As each group contains highly correlated features, every feature in a group doesn't need to keep. Keeping all of them increases the redundancy. So we depend on such a technique that can provide us an independent feature from each group. Principle Component Analysis is used to do this feature extraction.

- **Principle Component Analysis:** Principal component analysis PCA is a method for reducing the dimensionality of large datasets, increasing interpretability while minimizing information loss [19]. It accomplishes this by generating new uncorrelated variables keeping maximum variance. It is an orthogonal linear transformation that transforms data into a new coordinate system.

  In general, PCA operation is defined as:

$$\mathbf{Y} = \mathbf{XC}$$

  with $\mathbf{Y} \in R^{S \times P}$ is the projected data matrix containing P principal components of X with P $\leq$ N. The key is to find the projection matrix $\mathbf{C} \in R^{N \times P}$, that is equivalent to the eigenvectors of the covariance matrix of X, alternatively, to solve a singular value decomposition (SVD) problem for X[20],

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

  where $\mathbf{U} \in R^{S \times S}$ and $\mathbf{V} \in R^{N \times N}$ are the orthogonal matrices for the column and row of X, and $\sum$ is a diagonal matrix containing the singular values, λ_n, for $n = 0, \cdots, N-1$, non increasingly lying along the diagonal. The projection matrix C can be achieved from the first P columns of V with

$$\mathbf{V} = [\mathbf{v}_1, \cdots, \mathbf{v}_N]$$

and

$$\mathbf{C} = [\mathbf{c}_1, \cdots, \mathbf{c}_P]$$

where $\mathbf{v}_n \in R^{N\times 1}$ is n^{th} right singular vector of X, and $\mathbf{c}_n = \mathbf{v}_n$ The singular values contained in $\sum$ at $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ are the standard deviations of X along the principal directions in the space spanned by the columns of C [20]. λ_n^2 becomes the variance of X projection along the n^{th} principal component direction. Assumes, a component contributes how much information to data representation is measured by variance explained.

The PCA is applied to each group. From each group, one component as well as one feature is extracted. Later, the extracted features are concatenated with the lower correlative features. Lower correlative features are kept in their original form. In this way, the dimension of the feature set is reduced. As a result, an independent feature set gets ready to contribute to the machine learning model.

3.2.2 Classification Algorithm:

Classification algorithms are the main predictive part of a machine learning model. After getting a reduced feature set that has high independence, is used for training and testing the classification model. For classification we have used the below algorithm:

3.2.2.1 Naive Bayes Classifier:

Naive Bayes is a technique for classifiers using feature values. Naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given that the class feature. It works well in many real-world cases, such as document or text classification, spam filtering, and so on, and can be used for both binary and multi-class categories [21]. The NB classifier can be used to effectively identify the noisy instances in the data and create a robust prediction model. The main advantage is that, in comparison to more complex methods, it only requires a small amount of training data to rapidly estimate the necessary parameters. However, due to its strong assumptions on function

independence, its efficiency could be harmed. The most popular NB classifier variants are Gaussian, Bernoulli, Multinomial, Categorical, and Complement.

The generalized formula to calculate the probability of a data point x, belonging to a certain class c_i:

$$P(c_i \mid x) = \frac{P(x \mid c_i)P(c_i)}{P(x)}$$

3.2.2.2 Decision Tree Classifier:

Decision Tree is a supervised learning method that can be used to solve both classification and regression problems, but it is most often used to solve classification issues. DT classifies the instances by sorting down the tree from the root to a few leaf nodes. Starting at the root node, instances are identified by checking the attribute specified by that node, and then moving down the tree branch corresponding to the attribute value. The most widely used conditions for splitting are "gini" for the Gini impurity and "entropy" for the information gain, both of which can be represented mathematically as [21].

Entropy: $H(x) = -\sum_{i=1}^{n} p\left(x_i\right) \log_2 p\left(x_i\right)$

Gini: $(E) = 1 - \sum_{i=1}^{c} p_i^2$

3.2 shows a basic decision tree:

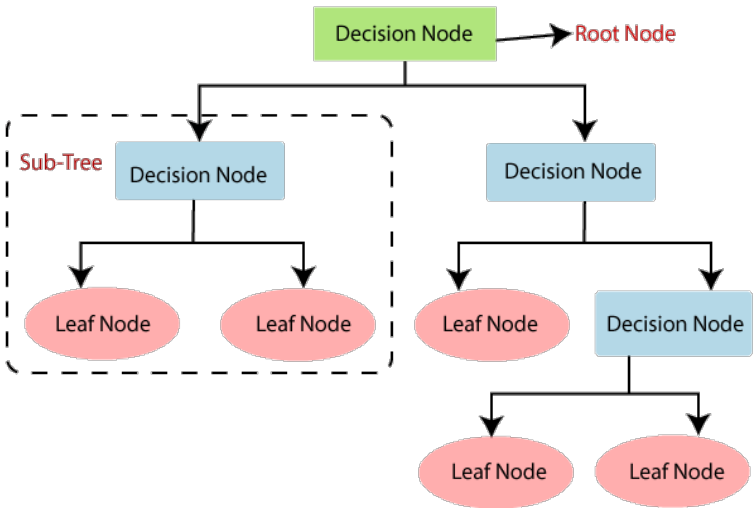


Figure 3.2: Decision Tree Structure

3.2.2.3 Random Forest Classifier:

A random forest classifier is a widely known ensemble classification methods used in machine learning for a variety of applications. This approach employs parallel ensembling, in which multiple decision tree classifiers are fitted in parallel on different data set sub-samples, and the outcome or final result is determined by majority voting or averages [21]. As a result, the over-fitting problem is reduced, and prediction accuracy and control are improved. As a result, the RF learning model with several decision trees is usually more accurate than a model based on a single decision tree. It combines a bootstrap aggregation (bagging) and random feature selection to create a set of decision trees with balanced variance. It can be used for both regression and classification problems.

3.3 shows a an example of a random forest structure considering multiple decision trees:

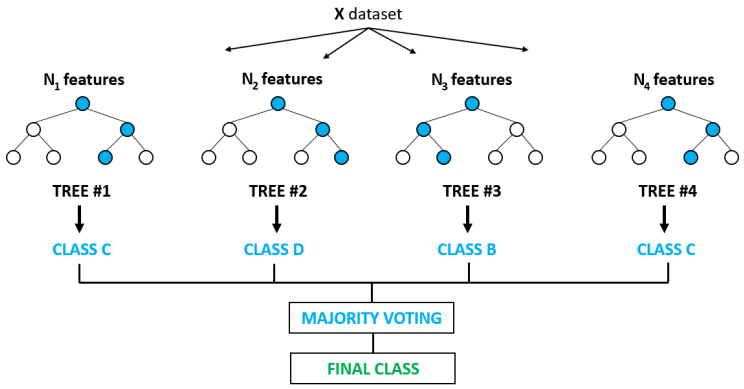


Figure 3.3: Random Forest Structure

3.3 Implementation

In this section, we discuss the implementation of the above described proposed methodology step by step.

3.3.1 System Requirements

To implement our approach required software and hardware tools are listed below.

3.3.1.1 Hardware Requirements

From preprocessing to prediction, our approach need the bellow major hardware tools.

- Intel core i5 CPU
- Minimum storage 80GB
- Main memory 8GB

3.3.1.2 Software Requirements

We implement our approach in specific software environment. They major items are listed here,

- Operating System : windows 10
- Python 3.7
- Anaconda framework
- Jupiter notebook

3.3.2 Raw Data Source

The source of our raw data is the server of the Canadian Institute of Cyber Security(CIC) [22]. The researchers of [4] installed about 5000 benign and 429 malware apps in three android devices. Then for each app, the app's behavior is captured connecting the laptop. App's behavior is captured three times: after installing, before reboot, and after reboot. We downloaded the captured data uploaded by CIC. In the captured data there are 8 behaviors for each app.

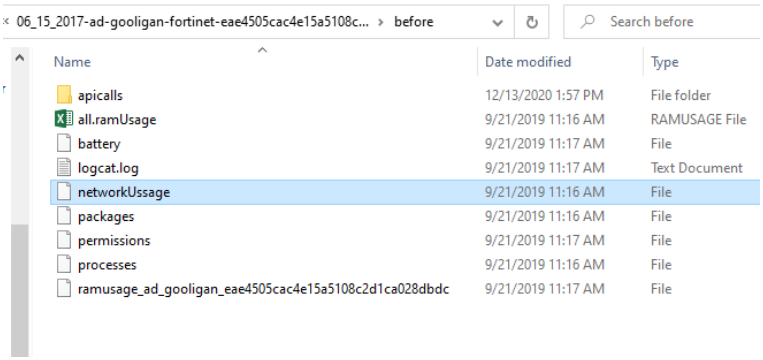


Figure 3.4: Inside of each application captured folder

Figure 3.4 shows that there is 8 unique behavior captured for each app. The downloaded data is about 50 GB in amount.

Among these behaviors, we have used the ram usage behavior. Converting the ram usage file in .txt format, we came to see the text data is in semi-structured form. So the text processing has been applied.

3.3.3 Text Processing

The data, ram usage file contains are semi-structured. Using regular expression the text data is converted into a structured CSV file.

ramusage_ad_Chinese_4dfb36ce42608ba7692540febfc97b48 - Notepad

File Edit Format View Help

```
Applications Memory Usage (kB):
Uptime: 1154337 Realtime: 1154327

** MEMINFO in pid 4507 [com.motor.mouse.multiple] **
                   Pss      Pss   Shared  Private   Shared  Private  Swapped     Heap     Heap     Heap
                 Total    Clean    Dirty    Dirty    Clean    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
  Native Heap     8861        0     1752     8784        0        0        0    16384    11356     5027
  Dalvik Heap    23271        0    17692    22948        0        0        0    46933    38839     8094
 Dalvik Other      624        0        8      624        0        0        0
        Stack      562        0       12      556        0        0        0
      Ashmem     2652        0        8     2640        0        8        0
     Gfx dev     6132        0        0     6132        0        0        0
   Other dev       56        0       40        0        0       56        0
    .so mmap     2904      800     1956      756    11416      800        0
    .apk mmap      53        0        0        0     1000        0        0
    .ttf mmap     101        0        0        0      556        0        0
    .dex mmap     457      176        0        0     1024      176        0
    .oat mmap     846       28        0        0    12536       28        0
    .art mmap    1543        0      996     1384     7052        0        0
   Other mmap     122        0        8        8      704       56        0
   EGL mtrack   13888        0        0    13888        0        0        0
    GL mtrack   13660        0        0    13660        0        0        0
      Unknown   16212        0       68    16212        0        0        0
        TOTAL   91944     1004    22540    87592    34288     1124        0    63317    50195    13121

 Dalvik Details
        .Heap     3404        0        0     3404        0        0        0
         .LOS    18252        0        0    18252        0        0        0
```

Figure 3.5: Ram usage data of a single app

In the Fig 3.5, the text are seen. We have used the Python OS module to apply

the string processing technique using regular expression. The first four lines are dropped. for some files, there is no data available in the text file. They are also detected through the regex.

In Fig 3.6, the left side red marked box along with the upper side box denotes the name of the feature of ram usage to make. Each element in the box is concatenated with others.

ramusage_ad_Chinese_4dfb36ce42608ba7692540febfc97b48 - Notepad

File Edit Format View Help

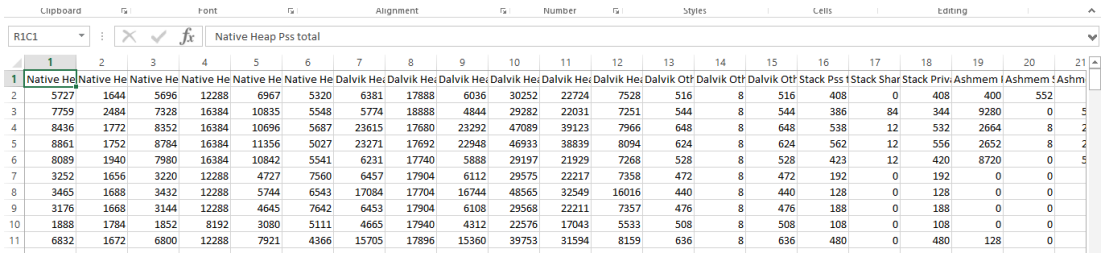
Applications Memory Usage (kB):
Uptime: 1154337 Realtime: 1154327

** MEMINFO in pid 4507 [com.motor.mouse.multiple] **

	Pss Total	Pss Clean	Shared Dirty	Private Dirty	Shared Clean	Private Clean	Swapped Dirty	Heap Size	Heap Alloc	Heap Free
Native Heap	8861	0	1752	8784	0	0	0	16384	11356	5027
Dalvik Heap	23271	0	17692	22948	0	0	0	46933	38839	8094
Dalvik Other	624	0	8	624	0	0	0			
Stack	562	0	12	556	0	0	0			
Ashmem	2652	0	8	2640	0	8	0			
Gfx dev	6132	0	0	6132	0	0	0			
Other dev	56	0	40	0	0	56	0			
.so mmap	2904	800	1956	756	11416	800	0			
.apk mmap	53	0	0	0	1000	0	0			
.ttf mmap	101	0	0	0	556	0	0			
.dex mmap	457	176	0	0	1024	176	0			
.oat mmap	846	28	0	0	12536	28	0			
.art mmap	1543	0	996	1384	7052	0	0			
Other mmap	122	0	8	8	704	56	0			
EGL mtrack	13888	0	0	13888	0	0	0			
GL mtrack	13660	0	0	13660	0	0	0			
Unknown	16212	0	68	16212	0	0	0			
TOTAL	91944	1004	22540	87592	34288	1124	0	63317	50195	13121
Dalvik Details										
.Heap	3404	0	0	3404	0	0	0			

Figure 3.6: Making features for single application

After getting the feature names, each line is then split and each numerical value is assigned for the features according to their position. For some features like Stack, Ashmen, .so mmap, .ttf mmap, .art mmap there is no value for the heap alloc, heap size, heap free. They are denoted as missing values. The structured data converted from 429 malware app's text files we get 3233 benign and 426 malware sample with 153 features in two separate CSV file which is shown in Fig 3.7. The class column is the family of the malware. The other three features named category and binary and md5 are also included for binary and category classification.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	Native He	Native He	Native He	Native He	Native He	Native He	Dalvik Hea	Dalvik Hea	Dalvik Hea	Dalvik Hea	Dalvik Hea	Dalvik Hea	Dalvik Oth	Dalvik Oth	Dalvik Oth	Stack Pss t	Stack Shar	Stack Priv	Ashmem I	Ashmem S	Ashm
2	5727	1644	5696	12288	6967	5320	6381	17888	6036	30252	22724	7528	516	8	516	408	0	408	400	552	
3	7759	2484	7328	16384	10835	5548	5774	18888	4844	29282	22031	7251	544	8	544	386	84	344	9280	0	5
4	8436	1772	8352	16384	10696	5687	23615	17680	23292	47089	39123	7966	648	8	648	538	12	532	2664	8	2
5	8861	1752	8784	16384	11356	5027	23271	17692	22948	46933	38839	8094	624	8	624	562	12	556	2652	8	2
6	8089	1940	7980	16384	10842	5541	6231	17740	5888	29197	21929	7268	528	8	528	423	12	420	8720	0	5
7	3252	1656	3220	12288	4727	7560	6457	17904	6112	29575	22217	7358	472	8	472	192	0	192	0	0	
8	3465	1688	3432	12288	5744	6543	17084	17704	16744	48565	32549	16016	440	8	440	128	0	128	0	0	
9	3176	1668	3144	12288	4645	7642	6453	17904	6108	29568	22211	7357	476	8	476	188	0	188	0	0	
10	1888	1784	1852	8192	3080	5111	4665	17940	4312	22576	17043	5533	508	8	508	108	0	108	0	0	
11	6832	1672	6800	12288	7921	4366	15705	17896	15360	39753	31594	8159	636	8	636	480	0	480	128	0	

Figure 3.7: CSV file containing processed structured data

3.3.3.1 Handling Missing, Constant and Duplicate Values

The features having all missing values are dropped straightforwardly. Few features have constant and quasi-constant values. Among them which are greater than 95% constant and quasi-constant are also dropped. The duplicate samples are removed considering the same value for each feature. By the end of this, we have got 89 features for 426 malware and 2775 benign samples.

3.3.4 Extracting Necessary Features

After getting the structured ram usage data, the dataset is pass through the Pearson Correlation. Applying Pearson Correlation we got the correlation matrix like Figure 3.8.

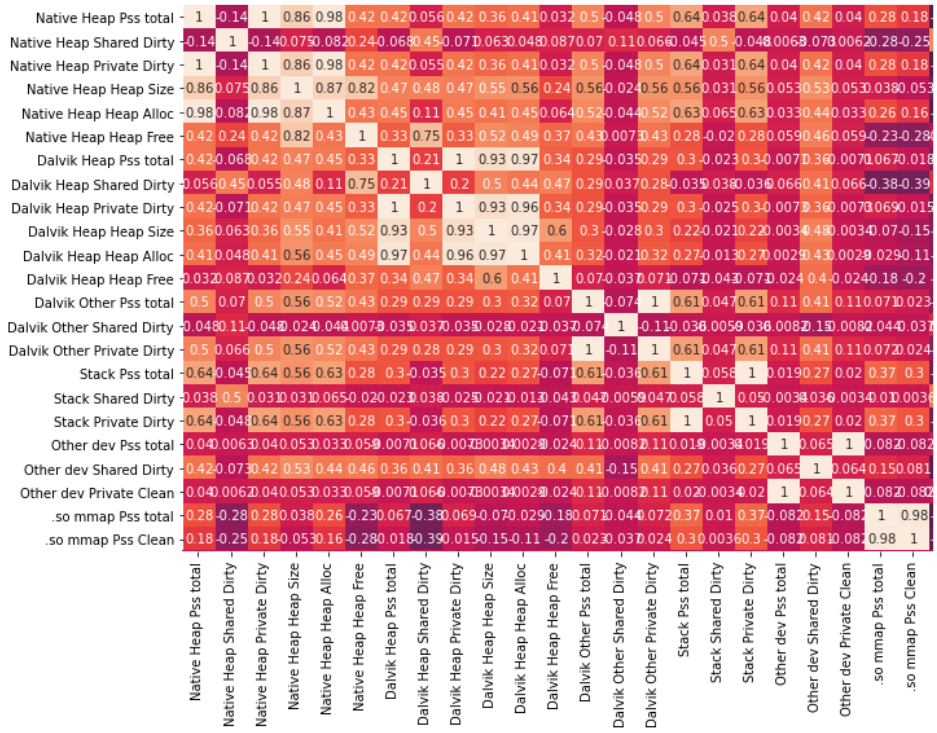


Figure 3.8: Partial heatmap snapshot of correlation matrix

From the correlation matrix, it is clear that some feature has a very strong correlation with one or more features. Which features are very correlated with others, is highlighted in Figure 3.9

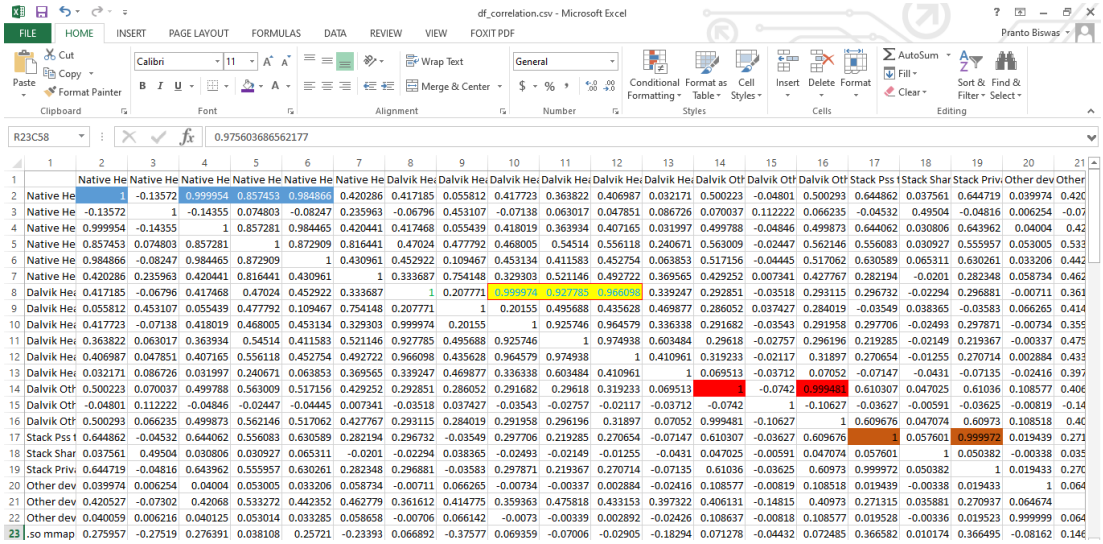


Figure 3.9: Correlatioin matrix

Group of Highly Correlated Features: To identify the groups, our proposed approach is applied. Here is the python code is stated:

Listing 3.1: Grouping method

```
grouped=[]
visited=[False]*90
for i in range(0,89):
  queue=[]
  queue.append(i)
  visited[i]=True
  col=[]
  while queue:
    q=queue.pop(0)
    col.append(q)
    for j in range(0,89):
      if q!=j and visited[j]==False:
        if cor_list[q][j]>.9 or cor_list[q][j]<-.9:
          #considering strong correlation
          queue.append(j)
          visited[j]=True
    #if len(col)>1:pair.append(col)
  if len(col)>1:
    grouped.append(col)
```

Applying our approach we have got 23 groups that contain 61 original features. The rest of the 28 features are not included in the groups due to the lower correlation value. In each group, the group members are shown similar behavior as their correlation value is greater than 0.9 or less than -0.9. The snapshot of the groups are listed in Figure 3.10

```
In [171]: grouped

Out[171]: [[0, 2, 4],
           [6, 8, 9, 10, 62, 61],
           [12, 14],
           [15, 17],
           [18, 20],
           [21, 22, 26, 56, 60],
           [25, 59],
           [27, 28, 30],
           [31, 32, 35],
           [37, 40],
           [41, 43, 44],
           [42, 46],
           [47, 51],
           [49, 71],
           [52, 54],
           [55, 58],
           [65, 67],
           [68, 70, 75, 76],
           [72, 78, 80],
           [73, 74],
           [77, 79],
           [82, 83, 85],
           [86, 88]]

In [172]: len(grouped)

Out[172]: 23
```

Figure 3.10: Grouped features

For each group, the group members are processed through Principal component Analysis (PCA) to transform into one feature. So for 23 groups we have got 23 features and we also have 28 original features. Concatenating the original 28 features with the extracted 23 features we have got the reduced 51 feature. This reduced feature set is used for model training.

3.3.5 Model Training

We have applied three different supervised classification algorithms on the reduced dataset. The Reduced dataset and the initial dataset both are applied in the classified for comparison purposes. We have used the 10 fold cross-validation to feed our data into model and prediction.

Chapter 4

Results and Discussions

4.1 Introduction

This chapter describes the experimental result of our working approach which is stated in the previous chapters. Our preprocessed reduced dataset is used for the evaluation using the classical machine learning model and after that, we have compared the result with other researchers' work.

4.2 Dataset Description

We have prepared two datasets containing the ram usage data of the android app. One dataset contains the benign app's information and another one is based on malware app's information. Two datasets are combined for binary classification. And for family and category classification malware dataset is considered. Table 4.1 shows information about the two datasets.

Table 4.1: Information of Benign and Malware Dataset

Dataset Name	**Number of Features**	**Number of Samples**
Benign_Dataset	51	2775
Malware_Dataset	51	376

Malware Dataset: The malware dataset has 376 samples. The malware samples are categorized into five categories and 40 families. Table 4.2 shows the family distribution of the malware app samples. Samples are fairly distributed in the families. Two small families are android.spy.277 and mobidash. 'android.spy.277' has 6 sample and 'mobidash' has 4 samples.

Table 4.2: Malware Family Names

	Class Name (Family)	Number of Samples		Class Name (Family)	Number of Samples
1	AndroidDefender	17	21	Jisut	10
2	Feiwo	12	22	Fakemart	10
3	Kemoge	11	23	RansomBO	10
4	Nandrobox	11	24	Gooligan	9
5	Jifake	10	25	Svpeng	9
6	Biige	10	26	Smssniffer	9
7	Zsone	10	27	Fakejoboffer	9
8	Simplocker	10	28	Beanbot	9
9	FakeApp.AL	10	29	Mazarbot	9
10	Fakeapp	10	30	FakeTaoBao	9
11	Fakenotify	10	31	Avpass	9
12	Ewind	10	32	Virusshield	9
13	Fakeinst	10	33	Dowgin	8
14	Wannalocker	10	34	Chinese	8
15	Fakeav	10	35	Plankton	7
16	Penetho	10	36	Lockerpin	7
17	Pletor	10	37	Porndroid	7
18	AvForAndroid	10	38	Youmi	7
19	Koler	10	39	Android.spy.277	6
20	Charger	10	40	Mobidash	4

4.3 Evaluation Measures

In order to evaluate our proposed system, we used several evaluation matrices. We will learn about these evaluation matrices in this section. Some of these evaluation matrices are,

- Confusion Matrix
- Precision
- Recall
- F1 score
- Accuracy

Confusion Matrix: A confusion matrix is a performance evaluation table for a classification model. For the binary classification model, the confusion matrix has two rows and two columns. This matrix reports the number of false positives, false negatives, true positives, and true negatives. As we have multiclass classification the confusion matrix is elaborated for each class with 40*40 size. For evaluation matrices, we need this table's values.

Table 4.3: Confusion Matrix

		Predicted class: P	Predicted class: N
Actual class	P	True Positives (TP)	False Negatives (FN)
	N	False Positive (FP)	True Negatives (TN)

- **True Positives (TP):** Samples where the true label is positive and class is correctly predicted to be positive.
- **False Positives (FP):** Samples where the true label is negative and class is incorrectly predicted to be positive.
- **True Negatives (TN):** Samples where the true label is negative and class is correctly predicted to be negative.
- **False Negatives (FN):** Samples where the true label is positive and class is incorrectly predicted to be negative.

  For multiclass classification, it is considered as a set of many binary classification problems.

Precision : Precision denotes as positive predictive value. That is the ratio of correctly classified positive samples to the total number of samples classified as positive. Precision can be achieved from the bellow equation.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

Recall : Recall is the ratio of correctly classified positive samples to the total number of positive samples. Also, it is called as true positive rate. Recall can be achieved from the bellow equation.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

High precision and high Recall is deserved for a good model. There is a trade-off between recall and precision, improving precision sometimes reduces recall and vice versa.

F1 score: A weighted harmonic mean of precision and recall that is normalized

between 0 and 1 is used to calculate the F1 score. Since precision and recall are inversely related, an F score of 1 indicates perfect balance. When both high recall and high precision are essential, a high F1 score is important. F1 score can be achieved from the bellow equation.

$$F1 = 2 \times \frac{\text{precision } \times \text{ recall}}{\text{precision } + \text{ recall}}$$

Accuracy: Number of samples correctly identified as truly positive or truly negative out of the total number of samples. Accuracy can be achieved from the bellow equation.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

4.4 Experimental Results

Through our approach, we have processed an initial dataset at first. Then the feature reduction has been performed on the initial dataset. So we have described here both the performance for the initial and reduced dataset for the classifiers. After that, We have compared our approach with the existing family classification method's result.

4.4.1 Result of Classifiers

Accuracy: Table 4.4 shows the accuracy of the different classifiers for the datasets having 89 features and 51 features respectively.

For every case, The optimized feature set gets a higher accuracy. As **random forest classifiers** has the highest accuracy, we have chosen this classifier to compare with other researcher's malware family classification methods.

The RF classifier has it's based on the feature importance selection technique. But if the feature set has many correlated features with each other then the redundant data doesn't impact the model. Moreover, for tree-based classification techniques the overfitting problem arises for redundant data. So while we have

Table 4.4: Accuracy of the classifiers on initial and reduced feature set

Classifiers	**Initial data set (89 features) (%)**	**Reduced feature set (51 features) (%)**
Naive Bayes	58.2	59.8
Decision Tree	58.4	59.8
Random Forest	70.4	72.7

removed the redundant data applying our feature reduction approach the tree-based classifier outperforms as there is lower confusion in decision making.

4.4.2 Result of Feature Extraction

In the next, Figure 4.1 we have compared our approach for feature reduction with PCA. The straightforward PCA with maximum accuracy at 63 features has less accuracy than our proposed approach.

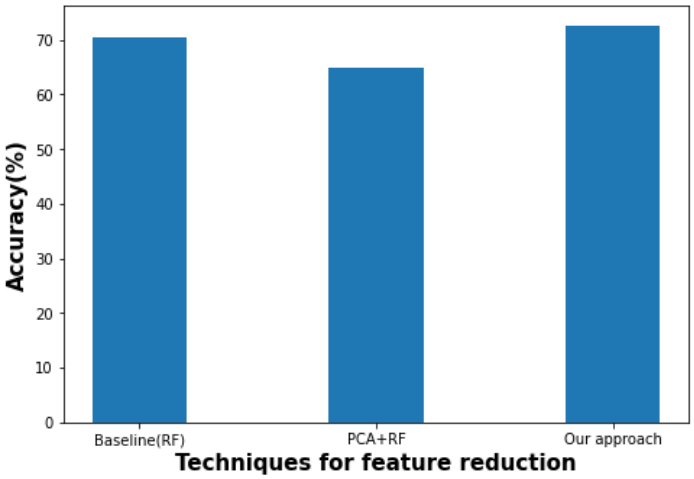


Figure 4.1: Performance comparison with PCA based feature reduction with our approach

As the straightforward PCA is an unsupervised feature reduction technique, the approach focuses on only reducing the dependency while information is missed in a bit. But in our approach, we consider the lower correlative features to exist in the reduced dataset. And for the higher correlative features, we have also kept much information using the PCA for the groups. That's why our approach contains much more information than the straightforward PCA technique Which helps in the accuracy gain.

4.4.3 Result of Malware Family Classification

The latest malware family classification based on real data is conducted by Taheri et al [4]. To compare with them we have also conducted the three specific classification binary, category, and family classification. The CICInvesAndMal2019 dataset contains 918 features for family classification and 8111 features for binary classification. Table 4.5 shows the result for precision comparison.

Table 4.5: Precision(%) comparison with CICInvesAndMal2019

	CICInvesAndMal2019	Our approach
Feature	API + Network traffic	Ram Usage
Binary	95.30	96.42
Category	83.30	91.40
Family	59.70	72.2

Table 4.6 represents the recall value comparison with the CICInvesAndMal2019 [4].

Table 4.6: Recall(%) comparison with CICInvesAndMal2019

	CICAndMal2019	Our approach
Feature	API + Network traffic	Ram Usage
Binary	95.30	93.65
Category	81.00	87.03
Family	61.20	72.90

For both of the cases our model outperformed for the family classification and also for the category classification. In comparison to the number of features, our dataset has only 51 features while CICInvesAndMal2019 contains 918 features. So, our approach is faster in the machine learning model. Below a bar chart represents the difference.

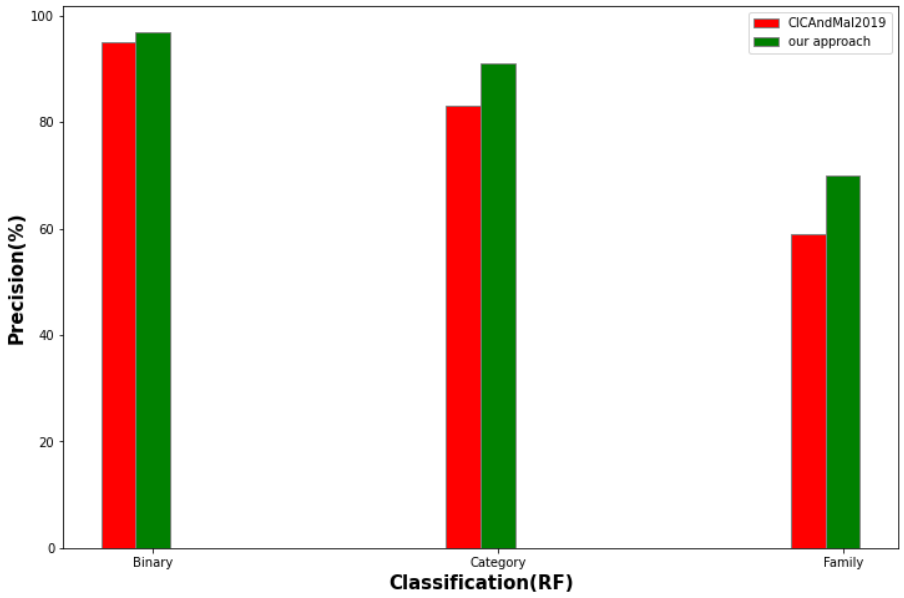


Figure 4.2: Comparision regarding precision between CICInvesAndMal2019 and Our approach

4.5 Discussion

From the experimental result, we can conclude that our approach of processing a ram usage dataset has got much more accuracy than the datasets for malware family classification provided by CIC. The dataset CICInvesAndMal2019, used by the CIC has the only binary value for each feature.

The reason behind the accuracy is the ram usage dataset contains the numerical value of various ranges. As the family number is 40 for our case, there needs an effective decision maker to classify. A different portion of ram such as heap, delvik, .jar provides a wide range of value for each malware category. That's why the ram usage data is effective for malware family classification.

The feature reduction technique helps to contain as much information as possible. Only the redundancy is removed in our approach using correlation and PCA applied in the correlative grouped features guarantees the very small loss of information.

The comparison between different classifiers explains that the feature reduction successfully helps the classifier to predict the test samples. In the below Fig 4.3 the members of 'jifake' and the 'beanbot' families are correctly identified. As

they are from the 'Premium SMS' category many of the characteristics are the same for them. But the .apk mmap shared clean feature which denotes the size occupied by apk code can separate them.

```
|   Native Heap Shared Dirty <= 1736
|   |   .apk mmap Pss Clean <= 4: fakenotify (9.0/1.0)
|   |   .apk mmap Pss Clean > 4
|   |   |   .so mmap Pss Clean <= 2220
|   |   |   |   .apk mmap Shared Clean <= 1076: jifake (10.0)
|   |   |   |   .apk mmap Shared Clean > 1076: beanbot (9.0)
|   |   |   .so mmap Pss Clean > 2220: zsone (2.0)
|   Native Heap Shared Dirty > 1736
```

Figure 4.3: Snapshot of a tree branches for family classification

Chapter 5

Conclusion

5.1 Conclusion

Malware detection is an important area in the security field of computing devices. It is not a concern of recent times. But detecting the malware needs different approaches when new operating systems are introduced. The approach source code scanning, network traffic checking, or other high-level feature analysis becomes obfuscated against the intelligent malware application[16]. But the malware keeps its every footprint in hardware level data which inspired us to find a memory usage pattern for the applications.

Despite the huge user of android operating devices, there was a lack of real smartphone ram usage data. We have provided a ram usage dataset of the real smartphone through our work to analyze finding the pattern of malware apps behavior. After detecting the app being malware, we have focused on the malware app's category and family classification so that the post-attack technique becomes easier for the security analyst.

Our proposed approach has found a most significant set of features from ram usage data to predict the family of the malware. The approach consists of removing the redundant data by analyzing correlations of the features and then extracting the best feature set for the model to get the very best result. To the best of our knowledge, there is no analysis and prediction is done on real smartphone ram usage data. Our technique will help to detect the anti-malware applications to take necessary steps quickly against the malware in real-time.

5.2 Future Work

Our proposed approach can be implemented on anti-malware applications to detect the malware families in real-time because of its low computational complexities.

The feature analysis has been done on 376 samples which were installed on two types of android versions. So after collecting samples from a variety range of android smartphones and analyzing them with our approach will improve the result.

Also, we have a plan to merge the ram usage feature set with other significant high or low-level data such as CPU usage, network traffic, API to get the best benefit.

References

[1] *Android apps on google play store*. [Online]. Available: `https://play.google.com/store/apps` (cit. on p. 1).

[2] Z. Qu, S. Alam, Y. Chen, X. Zhou, W. Hong and R. Riley, 'Dydroid: Measuring dynamic code loading and its security implications in android applications,' in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2017, pp. 415–426 (cit. on p. 1).

[3] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci and R. Baldoni, 'Android malware family classification based on resource consumption over time,' in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, IEEE, 2017, pp. 31–38 (cit. on pp. 1, 5).

[4] L. Taheri, A. F. A. Kadir and A. H. Lashkari, 'Extensible android malware detection and family classification using network-flows and api-calls,' in *2019 International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2019, pp. 1–8 (cit. on pp. 2, 5, 6, 15, 27).

[5] Y. Zhou and X. Jiang, 'Dissecting android malware: Characterization and evolution,' in *2012 IEEE symposium on security and privacy*, IEEE, 2012, pp. 95–109 (cit. on pp. 4, 6).

[6] F. Wei, Y. Li, S. Roy, X. Ou and W. Zhou, 'Deep ground truth analysis of current android malware,' in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2017, pp. 252–276 (cit. on p. 4).

[7] S. Türker and A. B. Can, 'Andmfc: Android malware family classification framework,' in *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, IEEE, 2019, pp. 1–6 (cit. on p. 4).

[8] T. Chakraborty, F. Pierazzi and V. Subrahmanian, 'Ec2: Ensemble clustering and classification for predicting android malware families,' *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 262–277, 2017 (cit. on p. 4).

[9] K. Aktas and S. Sen, 'Updroid: Updated android malware and its familial classification,' in *Nordic Conference on Secure IT Systems*, Springer, 2018, pp. 352–368 (cit. on p. 5).

[10] A. Martın, R. Lara-Cabrera and D. Camacho, 'Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset,' *Information Fusion*, vol. 52, pp. 128–142, 2019 (cit. on p. 5).

[11] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez and J. Blasco, 'Dendroid: A text mining approach to analyzing and classifying code structures in android malware families,' *Expert Systems with Applications*, vol. 41, no. 4, pp. 1104–1117, 2014 (cit. on p. 5).

[12] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto and L. Cavallaro, 'Droidsieve: Fast and accurate classification of obfuscated android malware,' in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 309–320 (cit. on p. 5).

[13] N. Kiss, J.-F. Lalande, M. Leslous and V. V. T. Tong, 'Kharon dataset: Android malware under a microscope,' in *The {LASER} Workshop: Learning from Authoritative Security Experiment Results ({LASER} 2016)*, 2016, pp. 1–12 (cit. on p. 5).

[14] A. H. Lashkari, A. F. A. Kadir, H. Gonzalez, K. F. Mbah and A. A. Ghorbani, 'Towards a network-based framework for android malware detection and characterization,' in *2017 15th Annual conference on privacy, security and trust (PST)*, IEEE, 2017, pp. 233–23 309 (cit. on p. 5).

[15] A. H. Lashkari, A. F. A. Kadir, L. Taheri and A. A. Ghorbani, 'Toward developing a systematic approach to generate benchmark android malware datasets and classification,' in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–7. DOI: 10.1109/CCST.2018.8585560 (cit. on p. 5).

[16] S. Banin and G. O. Dyrkolbotn, 'Multinomial malware classification via low-level features,' *Digital Investigation*, vol. 26, S107–S117, 2018 (cit. on pp. 5, 30).

[17] J. Milosevic, M. Malek and A. Ferrante, 'A friend or a foe? detecting malware using memory and cpu features.,' in *SECRYPT*, 2016, pp. 73–84 (cit. on p. 5).

[18] M. A. Hall, 'Correlation-based feature selection for machine learning,' 1999 (cit. on p. 10).

[19] I. T. Jolliffe and J. Cadima, 'Principal component analysis: A review and recent developments,' *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20 150 202, 2016 (cit. on p. 11).

[20] J. Zhang, 'Machine learning with feature selection using principal component analysis for malware detection: A case study,' *arXiv preprint arXiv:1902.03639*, 2019 (cit. on pp. 11, 12).

[21] I. H. Sarker, 'Machine learning: Algorithms, real-world applications and research directions,' *SN Computer Science*, vol. 2, no. 3, pp. 1–21, 2021 (cit. on pp. 12–14).

[22] *Cic-invesandmal2019 dataset*. [Online]. Available: https://www.unb.ca/cic/datasets/invesandmal2019.html (cit. on p. 15).