SIGNIFICANT FEATURES ANALYSIS FOR ANDROID MALWARE DETECTION USING MACHINE LEARNING TECHNIQUES

ASADULLAH HILL GALIB
Master of Science in Software Engineering
Institute of Information Technology
University of Dhaka
Class Roll: MSSE 0718
Session: 2018-19
Registration Number : 2014-016-466

A Thesis
Submitted to the Master of Science in Software Engineering Program Office
of the Institute of Information Technology, University of Dhaka
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

SIGNIFICANT FEATURES ANALYSIS FOR ANDROID MALWARE DETECTION USING MACHINE LEARNING TECHNIQUES

ASADULLAH HILL GALIB

Approved:

Signature	*Date*
Supervisor: Dr. B.M. Mainul Hossain	

To *Abdul Mannan Kajol and Khaleda Akter*, my parents whose endless inspiration has always kept me motivated

Abstract

Android malware is the key factor for the most security breaches in the android operating system. Malware authors are sharp-witted enough to evade the typical antiviruses or obsolete approaches of malware detection. As android malware generally tries to preserve the facade of a benign application using multifarious evasion techniques, it is worthy and necessary to take a perceptive approach to defend them. Detecting android malware effectively and feasibly in advance is the biggest challenge of this fast-growing digital world.

Typical malware detection techniques, like static analysis, dynamic analysis, hybrid analysis use full features set to classify malware. But the number of features is growing exceedingly with the growth of the Android system. These growing numbers of features make it complex and would misguide classifiers by over-fitting of data. So, rather than using full features set, analyzing significant features would help to reduce complexity as well as increase large scale malware detection. Several works [1, 2, 3, 4, 5, 6, 7, 8] suggest that significant features can adeptly classify malware. However, those works do not assess the performance of significant features with regard to the full features set.

This study aim at analyzing significant features for Android malware detection with consideration of maintaining performance. Permissions, API Calls, and ensemble features are analyzed distinctly in this study to assess the individual impact and overall performance of each type of feature. An approach is proposed in this study to analyze and identify significant features.

In the proposed approach, the features are incrementally analyzed using several feature selection techniques. According to the incremental feature selection a well-suited feature selection technique is selected and a minimal number of significant features are identified. Afterward, a correlation-based feature elimination strategy is applied for further reduction of significant features.

Experiments on two benchmark data sets show that the Recursive Feature Elimination with Random Forest Classifier causes the ~~the~~ minimal number of significant features in all cases. Evaluation results indicate that the proposed approach can notably reduce the features set. The reduced set of significant features can perform relatively close to the full set of features in terms of accuracy, recall, f-1 performance, AUC, and execution time. It also compares the performance of this approach with the related works and this work outperforms most of the works regarding malware detection rate. Furthermore, it reports the top significant features in malware detection. Finally, it implies that significant ensemble features are more effective than significant Permissions and API Calls. Also, significant API Calls perform better than significant Permissions.

This study signifies that significant features would be useful in classifying Android malware effectively while maintaining detection performance. These findings would accelerate large scale malware detection with consideration of performance.

Acknowledgments

I would like to appreciate my supervisor Dr. B.M. Mainul for his supportive guidance, continuous inspiration and candid advice. His support and help have inspired me to relentlessly work for producing the best quality work I can throughout the project. I would also like to thank my classmates of MSSE 7th Batch for their continuous motivation throughout the project.

List of Publications

1. Galib, A.H., Hossain, B. M. (2020, July). Significant API Calls in Android Malware Detection (Using Feature Selection Techniques and Correlation Based Feature Elimination). In Proceedings of the 32nd International Conference on Software Engineering Knowledge Engineering (pp. 566-571).

2. Galib, A.H., Hossain, B. M. (2019, December). A Systematic Review on Hybrid Analysis using Machine Learning for Android Malware Detection. International Conference on Innovation in Engineering and Technology (ICIET) 2019.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Android is the most prevalent mobile operating system (OS) currently: 72.23% of total mobile OS is Android [9]. Due to the openness, flexibility, and advancement of the Android platform, users, as well as developers of Android applications, are increasing day by day. All applications are not intended to serve users, malcontent developers create malevolent-intentioned malware applications.

Android malware is any kind of Android application which tries to exploit user's privacy, system features, or Android operating system. For instance, the malware tries to leak sensitive user data, disrupt normal functioning, take access controls, misuse permission, perform policy misconfiguration, etc. With the enormous growth of the Android system, Android malware also has grown significantly as well as upgraded its nature and activities [10]. On average 12,000 new malware instances are found per day [11]. To detect these diverse sets of malware, numerous features are used. These large features set not only increase complexity but also misguides the detection system. Thereby, identifying significant features for malware detection would be conducive. This study deals with analyzing significant features for malware detection.

The motivation behind the work has been addressed in this chapter. Also, this chapter includes the research questions and the contribution of this study. At the end of this chapter, a segment is listed on how the article is assembled.

1.1 Motivation

To defend against that malware phenomenon, researchers emphasized on Android malware detection to ensure Android mobile application security and user's privacy. Basically, researchers nowadays have focused on the machine learning approach for detecting Android malware, as the signature-based approach is being proved outdated by the malware authors. To detect Android malware, there are three approaches: Static Analysis, Dynamic Analysis, and Hybrid Analysis.

The static analysis uses the static features of the Android application such as Permissions, API Calls, etc. The dynamic analysis investigates the dynamic behavior of the application running on an emulated environment or on a real device. These dynamic features/behaviors include System Calls, Network Traffic, etc. Hybrid analysis tends to incorporate both the static and dynamic approaches into a common ground. Static and dynamic analyses have their own limitations. Currently, malware authors are too smart to evade these detection techniques. They use many evasion techniques to evade the analysis. For static analysis, commonly used evasion techniques by the malware authors are data obfuscation, control flow obfuscation, encryption, reflection, dynamically loaded code, repackaging, etc. [12]. For dynamic analysis, anti-analysis, mimicry, data obfuscation, misleading information flows and function in-directions, etc. are used as evasion techniques [12]. Besides, limited code coverage lessens the effectiveness of the dynamic analysis. The hybrid analysis approach integrates both static and dynamic analyses to mitigate their weaknesses. Though hybrid analysis is complex enough, it is effective and feasible according to related research.

However, all three approaches deal with a large number of features to classify malware using machine learning techniques that enhance sophistication. In all three approaches, many features are associated with each type of feature. For instance, 235 different Permissions are used in Android [13]. There are many features in all types of static and dynamic features. Using this large number of features would be problematic for malware detection. First, it causes computational complexity. Due to a large number of features, the detection process would be not efficient enough. Besides, it does not ensure effectiveness. Second, this large number of features would over-fit the machine learning model, which may engender the performances of Android malware detection. So, identifying significant features might be helpful regarding performance, effectiveness, and efficiency.

Among various features, Permissions and API Calls are the most used features in Android malware detection [14, 15, 16, 17, 1, 2, 3, 4, 5, 6, 7, 8]. Researchers perform malware detection mainly focusing on these two features, as most of the works have incorporated those features.

Android Permissions keep Android devices and their user safe. However, Android Permissions can be exploited by malware applications. Most notably, unconscious users may give Permissions to malware applications that should not be given. These problems leave room for the manipulation of malware. Hundreds of thousands of malware are now performing their malicious operations using Permissions exploitation. Additionally, a growing number of Permissions from Android platform updates make malware detection harder and more complex.

Android API (Application Programming Interface) is a series of specifications and guidelines that programs can follow to communicate with each other. Due to the wide-ranging applicability of API Calls, they are commonly used for characterizing and separating malware from benign applications. However, the Android operating system uses a large number of API Calls and the number continues to expand [18]. So, handling this large number of API calls in malware detection for Android is challenging.

These large numbers of Permissions and API Calls would overfit the classifier model or complicated the classification method by providing a large number of features set. It would be useful to boost this problem by reducing features while detecting malware. Several studies [1, 2, 3, 4, 5, 6, 7, 8] also hints that analyzing significant features can reduce complexity. However, those related works lack conclusiveness. Those studies only deal with reducing the number of features but do not maintain or analyze performance. So, the aim of this study is to analyze significant features as well as to maintain performance in Android malware detection using machine learning techniques.

1.2 Research Questions

Most of the existing research mostly work on static analysis, dynamic analysis, and hybrid analysis of Android malware detection. A few works deal with significant or important features for Android malware detection. However, a few works work essentially on reducing the number of features and suggesting complexity reduction methods. They do not assess the performance of reduced features with regards to considering all features. But whether or not they compromise performance is hard to determine.

As the related works mostly overlooked the importance of significant features identification and ignored analysis of performance, this work tends to contribute in this regard. This works aims to answer the following research question.

- **RQ**: *How can we detect Android malware using significant features based on machine learning techniques?*
  To detect Android malware effectively and efficiently, analyzing significant features would be beneficial as suggested by many research [1, 2, 3, 4, 5, 6, 7, 8]. This question can be answered more precisely by investigating the following sub-questions.
  - **SQ1**: *How can we select the significant features for Android malware*

detection using machine learning techniques?

By answering this question, a strategy can be defined for selecting significant features from the extracted features automatically. Finding out the effective combination of significant features is the primary goal of this question. Also, the best-suited feature selection algorithm in this regard and the number of features to be selected can be figured out while answering this question.

– **SQ2**: *How do the significant features perform in Android malware detection?*

  Identifying significant features would help to build an effective model, which will lead to better malware detection. The run-time performances of the significant features would indicate the efficacy and applicability of it. If significant features do not perform similar to all features, it will not be effective in Android malware detection. So, the primary concern here is whether significant features perform proficiently with respect to all features. With regard to performance analysis, several performance measures should be considered for robustness. For instance, detection accuracy, precision, recall, execution time, etc. should be analyzed for assessing performance. A comparison of findings with existing works would also help explain this studyś significance and viability.

By answering this question, significant features can be analyzed and deduced for Android malware detection. It would also enable a new research direction.

1.3 Contribution

This work leads to the study and identification of significant features for the detection of Android malware. From Section 1.2, it is clear that there are primarily two objectives that suggest an approach to effectively identify significant features for Android malware detection and evaluate the performance of significant features in Android malware detection.

Addressing the first objective, an approach is proposed to analyze significant features in Android malware detection. The approach analyzes the best-suited feature selection algorithm and selects a minimal range of significant features in malware detection. After feature extraction and data pre-processing, an incremental feature selection (IFS) process is carried out for several feature selection algorithms, such as Mutual Information Gain, Univariate ROC-AUC scores, Recursive Feature Elimination (RFE) with Gradient Boosting Classifier, and Random Forest Classifier, SelectKBest using chi scoring function, SelectFromModel using Random Forest and Extra Trees classifiers. IFS incrementally select 1 to n number of features (where, n = total number of features) using those feature selection techniques and evaluate performances for all the features. Self-descriptive plots of performance metrics from the IFS assist to determine the best-suited feature selection algorithm and the minimal range of significant features. The minimal range of features denotes the range of features from which the performance results do not improve despite the number of features increased. Afterward, a correlation-based feature elimination is implemented to further reduce the defined minimal range of features. A less important feature from each of the highly correlated pairs (with respect to the target class) is omitted as it does not compromise performance.

To address the second objective, the performance of the significant features are evaluated on two benchmark data sets separately, such as Drebin and Android Malware Genome Project (Malgenome). Evaluation of two data sets separately is due to ensure generality. Using the final minimal range of features and the nominated feature selection algorithm, performance evaluation is carried out from three perspectives. Firstly, the detection performance is evaluated using accuracy, precision, recall, f1 score, and ROC-AUC of significant features with respect to all features. Secondly, the run time performance is assessed using execution time.

Thirdly, related works are compared in terms of performance measure (accuracy, precision, recall, f1 score).

Significant features are analyzed in the following three aspects separately:

1. Significant API Calls Analysis
2. Significant Permissions Analysis
3. Significant Ensemble Features Analysis

The rationale behind this distinct analysis is to analyze the significance of individual types of features: Permissions and API Calls. This study also seeks to assess whether or not the ensemble features work better than individual feature types.

Experimental results show that the proposed approach can effectively identify significant features by reducing features. For the Drebin dataset, it reduces 73 API Calls to 17-21 API Calls, 114 Permissions to 33-37 Permissions, and 187 ensemble features to 25-30 features. For the Malgenome dataset, it reduces 69 API Calls to 15-21 API Calls, 113 Permissions to 22-26 Permissions, and 184 ensemble features to 12-15 features. Recursive Feature Elimination (RFE) with the Random Forest classifier turns to be the best-suited feature selection algorithm for all cases. However, RFE with Gradient Boosting, Feature Importance using Random Forest, and Extra Trees also perform well.

Moreover, significant features not only decrease features and complexity but also retains performance as similar to all features. For instance, using only the top 13 (out of 184) significant ensemble features, the performance metrics are as follows for the Malgenome data set: accuracy - 97.32%, precision - 96.58%, recall - 95.78%, f1 score - 0.961, and AUC – 0.994. For the same data set, using all the 184 ensemble features, the performance metrics are as follows: accuracy - 98.32%, precision - 98.95%, recall - 96.24%, f1 score - 0.975, and AUC – 0.998. Thereby, significant ensemble features can remarkably reduce features set while keeping performance consistent. Both significant API Calls and significant Permissions yield similar results.

Likewise, as far as the execution time is concerned, the significant features take fairly less time. For instance, using only the top 13 (out of 184) significant ensemble features, the execution time is 3.21 seconds while the full features set (184 features) take 16.6 seconds. Besides, in comparison with existing works, this study outperforms almost all of the related studies while using fewer numbers of significant features. Finally, the top significant API Calls, Permissions, and ensemble features are reported.

Lastly, this study suggests that significant ensemble features are most effective in malware detection than significant Permissions and API Calls. This finding is also supported by Aswini et al. [1]. Apart from that, significant API Calls are more effectual than significant Permissions in detecting malware detection.

All in all, this study succeeds in analyzing significant features for Android malware detection while preserving performance in detection. It allows the approach to be implemented on a broad scale with a viable strategy.

To be specific, this work makes the following contributions:

1. It proposes and assesses an approach for analyzing and identifying significant features in Android Malware Detection effectively.

2. Its reduced significant features perform notably with respect to the full features set in terms of accuracy, precision, recall, f-1 score, AUC, and execution time. Besides, it outperforms most of the related works in detection performance.

3. It provides a list of top significant API Calls, significant Permissions, and significant ensemble features in Android malware detection.

4. It provokes a discussion about the challenges and opportunities of analyzing significant features as well as Android malware detection.

1.4 Organization of the Report

In this section, the report organization has been shown to provide this document with a road-map. In the following, the organization of the chapters in this study is listed.

Chapter 2: The definitions and background information about Android malware detection have been discussed in this chapter.

Chapter 3: In this chapter, the related works for significant features in Android malware detection have been presented in a structural way. Also, existing works on static analysis, dynamic analysis, and hybrid analysis have been discussed.

Chapter 4: This chapter contains the proposed approach for analyzing significant features in Android malware detection.

Chapter 5: The experimental setup, implementation, and evaluation based on three aspects have been provided in this chapter.

Chapter 6: This is the chapter that summarizes the whole report and highlights future work.

Chapter 2

Background Study

With the growing complexity of the Android system and uncertainty in the environment, Android malware is also evolving tremendously in terms of nature, behavior, and evasion techniques [19, 20]. To protect against this phenomenon of malware, researchers stressed Android malware detection to ensure the security of Android mobile apps. Researchers mostly rely on machine learning-based malware detection using static and dynamic features. However, the increasing number of features and corresponding complexities make the detection process difficult in terms of model training as well as classification. So, identification of significant features can reduce the overhead of a large number of features while maintaining the performance of malware detection.

2.1 Android Malware

Android malware is an application running on the Android OS that implicitly or explicitly performs malicious activities. It includes virus, worm, trojen, rootkit, adware, ransomware, spyware, and other malicious application. It can cause threats from multifarious directions [21], such as:

- disrupt normal functioning

- leak information
- take access controls
- root exploitation
- manipulate data
- expose private content
- disrupt services
- phishing
- permission misuse
- policy misconfiguration
- incorrect input validation

Moreover, malware is growing exceedingly to keep pace with the immense growth of Android applications. In each month, on average almost 10 million new malware is introduced [9]. New malware is found in every 10 seconds [10]. The most alarming fact is that nowadays noxious malware authors also aware of the malware detection system and they use many novels and crafty evasion techniques to avoid detection. Some common evasion techniques [12] by malware author are given below:

- Data obfuscation
- Control flow obfuscation
- Dynamically loaded code
- Anti-analysis
- VM-aware

- Reflection
- Encryption

Apart from these common evasion techniques, malware authors try to invoke new techniques. Researchers also deal with these issues using cutting-edge technologies and effective malware detection approaches.

2.2 Android Permissions

Android Permissions guards the security of the Android system and its user. When a user installs an app, it will request certain Permissions to move on. In order to use particular privileges or access sensitive data like sending SMS, opening camera, recording audio, location, etc., applications need approval from the user and the system [22]. Sometimes, the Android system automatically grants Permissions to the app when there are not any chances of Permissions exploitation or in case of very common and basic Permissions. Most of the cases, the Android system prompts the user to grant the Permissions. According to the user's necessity and requirement consideration, the user can grant or deny Permissions. By doing so, the Android system builds a safeguard using automated as well as user intervention.

A key design point for the Android security architecture is that by default no application has the authorization to perform any activities that will adversely affect other applications, the operating system, or the user [22]. For instance, one application can not get or manipulate data from other applications without any Permission. Also, one application can not get access to the userś sensitive data or can not control system features by default. This architecture ensures privacy and security for users as well as for the Android system.

However, there are many loopholes that might be exploited by the malicious malware. Most commonly, unaware user can grant Permissions to malicious applications which should not be granted. These issues leave space for malware apps to exploit. Thereby, hundred of thousands of malware apps are still executing their malicious activities and various types of malware are increasing excessively.

Android Permissions play important role in Android malware analysis because malware requires certain Permission to perform a series of malicious activities [23]. Some of the frequently abused Permissions are *SEND_SMS, READ_SMS. WRITE_SMS, USE_CREDENTIALS, GET_ACCOUNTS, MANAGE_ACCOUNTS, RECORD_AUDIO*, etc. Developers suggest several recommendations while dealing with Android Permissions [24], such as:

- Using just the Permissions the app requires to work with
- Contemplating Permissions required by libraries
- Being unambiguous and clear-cut about Permission usage and system accesses

Maintaining these recommendations as well as awareness of users would help to prevent malware exploitation.

2.3 Android API Calls

Android API (Application Programming Interface) is a series of rules and standards that can be implemented by applications to communicate with each other [18]. This communication happens through API Calls. API Calls are the building block of internal and external communication. API Calls are frequently used to connect with external resources or platforms. For instance, when a user needs to check data from an external database or server, his request is being invoked by a set of API Calls. After the request processing, API Calls are again used to receive data from the external sources and make it available for the user.

Android application developers also take API levels into account, for many reasons in particular. To implement forward and backward compatibility of Android applications, API levels are used. Nearly all modifications to the API levels are additive. All application is forward-compatible to later releases; API levels are not supposed to be expired [25].

Android APIs are now frequently modified and improved with the massive growth of the Android platform. Every consecutive update of the Android platform brings noteworthy changes to the APIs it offers [26]. In the latest Android platform version (Android 11), the API level is 30.

API Calls are widely used for analyzing and distinguishing malware from benign applications since API Calls are widely used. According to existing research, malware applications tend to use a certain set of API Calls while exploiting. API Calls are more conducive to malware detection than Permissions because many malware does not need any dangerous Permissions, but several sets of API Calls need to be used to interact and exploit within the Android system.

2.4 Detection Techniques

Researchers generally analyze Android malware with the following three approaches: Static Analysis, Dynamic Analysis, and Hybrid Analysis. Each approach has individual strengths and weaknesses.

2.4.1 Static Analysis

In static analysis, various static features are extracted from source code and metadata. If the source code is not available, reverse engineering is applied to reproduce the source code. According to the static features, a detection model is built using machine learning techniques to classify Android malware.

Researchers used Androguard, ApkTool, Appknox, DroidMat, etc. tools for static analysis. According to the existing research in the static analysis, the most used static features are Permissions and API Calls. The common static features [21] used in the literature for static analysis are listed below:

1. Permissions
2. API Calls
3. Intents
4. Instructions/Commands
5. Hardware Usage Analysis
6. Meta-data
7. Signatures
8. Control Flow Graph
9. Suspicious Files
10. Potentially Dangerous Functions and Methods

Static Analysis faces many troubles such as data obfuscation, control flow obfuscation, encryption, reflection, dynamically loaded code, repackaging, etc. by the shrewd malware authors [12].

2.4.2 Dynamic Analysis

The dynamic analysis deals with the dynamic features/behaviors of an application. To track the dynamic behaviors of an application, the application is to be run/executed in an emulated environment or on a real device. A detection model is also built here according to the dynamic features.

Researchers used Droidbox, Marvin, Cuckoo Sandbox, AppsPlayground, DroidLogger, etc. tools for dynamic analysis. According to the existing research in dynamic analysis, the most used dynamic features are System Calls. The common static features [21] used in the literature for dynamic analysis are listed below:

1. System Calls
2. Network Traffic
3. Running Services
4. File Operations
5. Phone Events
6. Network Operations
7. CPU Consumption
8. Memory Consumption
9. Binder

On the other hand, Dynamic Analysis also has some drawbacks. To evade dynamic analysis, the anti-analysis technique is used frequently by malware authors to detect virtual machines or emulated environments. If the application detects emulated environments in advance, they will act as a benign application. By doing so, the dynamic analysis might fail to detect Android malware. Besides, malware authors use mimicry, data obfuscation, misleading information flows, and function indirections, etc. to evade dynamic analysis [4]. The biggest weakness of dynamic analysis is limited code coverage: covering all paths is not feasible when investigating the dynamic behavior of an application.

2.4.3 Hybrid Analysis

Hybrid Analysis incorporates both static and dynamic features for detecting Android malware. As it deals with both static and dynamic features, it is computationally more complex. Andrubis, AndroData, etc. are used by the researchers for hybrid analysis.

The hybrid analysis integrates both static and dynamic features for effectiveness. Firstly, it seeks to extract the static and dynamic features of Android applications. After that, those extracted static and dynamic features are combined to build a detection model. Finally, according to the static and dynamic features, a detection model is built using machine learning techniques to classify Android malware.

By incorporating static and dynamic approaches into a common ground, the hybrid analysis leads to more complexity in Android malware detection. The detection process is more likely to take more time and effort. Though the hybrid approach might be more effective for Android malware detection than the static or dynamic approach, accomplishing a viable malware detection technique is challenging. As the hybrid approach is the combination of static and dynamic approaches, this approach can overcome the individual weakness as well as can accumulate the advantages of them. Thereby, the hybrid approach strengthens the detection process with the cost of time and complexity. Hybrid methods can also increase robustness, monitor edited apps, increase code coverage, and find vulnerabilities effectively[4].

2.5 Summary

With the major use of the Android system, malware exploitation is also increasing. So, Android malware detection becomes a widely studied topic. The technical areas of Android malware detection were examined, and approaches from various types were explored from a birdś-eye view. These domains shape the scope and depth of this subject of study. In this chapter, the basic ideas and concepts of Android malware detection are discussed. Also, several generic detection techniques and important features of malware detection are discussed. This discussion can help to understand the existing works in self-adaptive system design.

Chapter 3

Literature Review

In this chapter, the existing Android malware detection approaches will be discussed. In the literature, numerous detection approaches have been proposed. In particular, works concerned with the analysis of important features for malware detection are explored in depth. Most of the works analyze malware using static, dynamic, or hybrid analysis approach. Though this study deals with identifying significant features, the common approaches are also discussed briefly.

Although there are many attempts to detect malware using static and dynamic analyzes, many of them do not use machine learning techniques. This chapter focuses primarily on machine learning related malware identification efforts.

From the literature mentioned below, six types of Android malware detection are evident depending on the form of strategies adopted by these.

1. Significant Ensemble Features Based Approaches
2. Significant API Calls Based Approaches
3. Significant Permissions Based Approaches
4. Static Analysis Based Approaches
5. Dynamic Analysis Based Approaches

6. Hybrid Analysis Based Approaches

The first three ones are closely related to this study as they deal with significant or important features for Android malware detection. The first approach which is the closest to this work deals with API Calls and Permissions features like this study. To the best of our knowledge, only one prior work is found belongs to this approach. The later three approaches are most common in Android malware detection. In the remaining sections, all these approaches are discussed.

3.1 Significant Ensemble Features Based Approaches

Though a handful number of works employed API Calls and Permissions, only two work deal with identifying important API Calls and Permission at a time.

Optimal Ensemble Features

Aswini et al. [1] compares the significance of ensemble features with regards to independent features in Android malware detection. In order to identify significant features, five feature selection techniques are evaluated, such as Bi–Normal Separation, Mutual Information, Relevancy score, Kolmogorov dependence, and Kullback Leibler. AdaBoostM1 with J48, Random Forest, and J48 is used for the classifier model. Permissions and API Calls are assessed incrementally to identify optimal features set.

They experimented with 1175 apps including 600 benign and 575 malware apps where the malware samples are collected from Contagiodump. They considered Permissions, Permission count, hardware features, software features, and API Calls as feature categories. Ensemble features are developed by combining the optimal features from the feature categories.

From the experimental results, 30 Permissions derived from the Bi–Normal Separation provides the highest accuracy of 92.51% while using the Random Forest classifier in Permissions analysis. In API Calls analysis, 100 API Calls resulting from the Bi–Normal Separation have the best - 91.83% accuracy when using Random Forest Classifier. But ensemble features combining Permissions and API Calls outperforms the independent feature categories. While using 168 ensemble features (boolean) with the Bi–Normal Separation, it achieves the highest accuracy of 93.02% with the Random Forest classifier.

These works suggest that ensemble features comprising the optimal features can classify Android malware more accurately rather than using Permissions or API Calls features independently. Also, the Bi –Normal Separation outperforms other feature selection techniques and Random Forest Classifier outperforms other classifiers in malware detection. Moreover, increasing the number of features includes irrelevant features that decrease the accuracy of malware detection.

Fest

Zhao et al. [2] analyze top features using their proposed feature selection algorithm - FrequenSel. They use Permissions, API Calls, Actions, IP and URL features and identify top features from them. They compare the feature selection techniques Chi-Square and Information Gain and examine that these two feature selection algorithms have drawbacks in this regard like distribution bias and long-tail effect. They analyze top features considering the frequency of features belongs to malware apps with respect to benign apps.

They evaluate their approach using 3986 malware apps from Drebin, and 3986 benign apps from PlayStore. They classify malware for different features using Support Vector Machine (SVM), KNN, J48, and Naive Bayes classifiers. The FrequenSel algorithm generates four subsets of features containing 241, 262, 309, 398 features. The features set are then ranked using the Chi-Square and Information Gain algorithm. The top features are analyzed according to those two feature selection techniques and they show that their proposed algorithm - FrequenSel can identify top features for malware detection.

Finally, they evaluate the performance of malware detection using the top 398 features derived from their approach. Experimental shows that they achieve 97.5% accuracy with 97.5% recall, 3.2% false-positive rate, and 97.5% true positive rate while using an SVM classifier.

3.2 Significant API Calls Based Approaches

Some works analyze important or critical API Calls in Android malware detection though they overlook other features. These works are discussed in this subsection thoroughly.

DroidAPIMiner

DroidAPIMiner [3], proposed by Aafer et al. analyzes critical API Calls in Android malware detection using frequency analysis. It shows that API Calls based features outperformed Permission based features. It extracts dangerous APIs, package level information, and APIs parameters. It refines its feature sets considering the support of APIs, third-party invoking APIs, and data flow analysis. It evaluates their approach using 20,000 apps including 3987 malware apps. Malware apps are collected from McAfee and Android Malware Genome Project. It trains the model with a KNN classifier and assesses its performance using accuracy, TPR, and TNR.

It evaluates the performance of the top 10, 40, 80, 120, and 169 API Calls. Also, it compares the performance with the top 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, and 124 Permissions. Experimental results indicate that API Calls based features perform better. While using 169 API Calls with the top 20 used parameters, it gains the highest accuracy of 99% with a TPR of 97.8%. Apart from the performance, it takes 25 seconds on average to detect Android malware.

Correlation Based Sensitive API Calls

Zhao et al. [4] identifies sensitive API Calls to detect Android malware using mutual information-based correlation. The correlation between API Calls and Android malware is measured using mutual information and it gives a sensitive score between 0 to 1. They demonstrate that taking the top 20 sensitive API Calls is the most convenient as the performance does not increase if more API Calls are taken into consideration.

At first, they extract the API Calls and generates a 20-dimensional eigenvector. Afterward, it employs an ensemble learning model using KNN and decision tree algorithm for model training and defines a voting strategy to classify malware. In this work, 528 malware samples and 516 benign samples are used for training and 100 apps are taken for testing purposes. Malware samples are taken from virusShare.com. According to experimental results, their approach achieves 92% accuracy with a precision of 93% and TPR of 89% while gives a high FPR.

Grouping of API Calls

Alazab et al. [5] proposes grouping strategies for selecting important API Calls in Android malware detection. They consider the occurrences of API Calls in benign and malware apps. Using set-theoretical intersection and the occurrences of API Calls, their strategy divides the API Calls into three distinct groups as follows:

1. **Ambiguous group:** The intersection of API Calls which are commonly utilized by benign and malware apps.

2. **Risky group:** The intersection of API Calls which are used by malware apps more frequently than benign apps.

3. **Disruptive group:** The intersection of API Calls which are frequently used by malware apps but not by benign apps.

They aim to classify malware using the API Calls from the risky group and disruptive group. They also analyze the feature importance of API Calls using the Mutual Information Gain algorithm to justify their grouping. According to the feature importance, the top 12 important features in malware detection belongs to the risky group.

In their work, it uses 13,719 malware samples collected from Androzoo, Contagio, MalShare, VirusShare, and VirusTotal. Also, it uses 14,172 benign samples collected from PlayStore. They employ Random Forest, J48, Random Tree, KNN, Naive Bayes classifiers for evaluation. Using 1326 API Calls from the risky group and the disruptive group, their approach achieves an F-measure of 94.3% with Random Forest classifier.

3.3 Significant Permissions Based Approaches

Apart from the important API calls, there are other works in Android malware detection that evaluate important Permissions even if they ignore other features like API Calls. These works are discussed in this subsection thoroughly.

SigPID

Sun et al.[6] proposes a 3-level data pruning approach to identify significant Permissions in Android malware detection. They successfully reduce 135 Permission features to 22 features while maintaining performance. Their 3-level data pruning methods are described as follows:

1. **Permission Ranking with Negative Rate (PRNR):** It gives a conclusive ranking of Permissions considering the usage of Permission by benign and malware apps. For all the Permissions, their support corresponds to the benign, and the malware dataset is calculated. Afterward, the PRNR of the Permissions is calculated using the supports. PRNR ranges from -1 to 1. PRNR of -1 indicates that the Permission is only used by the benign dataset, PRNR of 1 indicates the Permission is only used by the malware dataset, and PRNR of 0 indicates that the Permission impacts very little in Android

malware detection. After applying PRNR, they reduce 135 Permissions to 95 Permissions.

2. **Support Based Permission Ranking (SPR):** Here, it uses the support of each Permission as exclusion criteria. If the support of the Permission is low, it is excluded from the significant Permissions for little impact. After applying SPR, they reduce 95 to 25 Permissions.

3. **Permission Mining with Association Rules (PMAR):** Finally, they exclude one of the permissions which appear together in an app. To identify it, it uses PMAR and reduces 25 to 22 Permissions.

They use 5494 benign apps and 1661 malware apps for evaluation. They evaluated the five best classifiers from 67 classifiers. And, Functional Tree is the optimal classifier for analyzing malware using significant Permissions. Experimental results show that using the significant 22 Permissions and the optimal machine learning algorithm, they achieve performances, such that accuracy - 95.63%, precision - 97.54%, recall - 93.62%, FPR - 2.36%, F-measure - 95.54%. Moreover, they demonstrate that significant Permissions in particular takes less training and testing time.

Their reduced number of Permission features have a higher recall value, close enough accuracy value with the full features set. But their precision was lower and false positive rate (FPR) was higher notably with respect to all Permission features.

ANFIS-FCM using Significant Permissions

Altaher et al.[7] proposes an approach based on an adaptive neuro-fuzzy inference system (ANFIS) with fuzzy c-means clustering using significant application permissions. In Permissions analysis, they use the Mutual Information Gain algorithm for feature selection. The top 24 ranked features according to feature importance are taken for classification.

They use the Android Malgenome Project dataset to collect malware samples and uses PlayStore to collect benign samples. Their classification accuracy is 91%, with a False Positive Rate (FPR) of 0.5% and a False Negative Rate (FNR) of and 0.4%.

Risky Permissions using Feature Ranking and Subset Selection:

Wang et al.[8] explores the Permission-induced risk in Android applications. In the first step, they evaluate individual permissions and collective permissions and implemented three measures of scoring on the permission features. To identify the risky Permissions, the following three feature ranking methods are used: mutual information, correlation coefficient, and T-test. Afterward, they discover risky permission subsets using Sequential Forward Selection (SFS) and Principal Component Analysis (PCA). Moreover, they assess the risky Permissions using Decision Tree, Random Forest, and Support Vector Machine classifiers.

Using the ranking methods they report the top 40 risky Permissions ranked by the three methods separately. In order to obtain the risky Permission subsets, three ways are suggested as follows:

1. Selecting top-k Permissions according to the risky Permissions ranking
2. Applying Sequential Forward Selection (SFS) to select top k permission sets
3. Applying Principal Component Analysis (PCA) to select top k principal components

They assess all three ways with the classifiers. They use 3417 malware apps and 310,926 benign apps in their study. The experimental result shows that using the 40 top risky Permissions, all the methods including Decision Tree and Random Forest gain 92.79% detection with high TPR and low FPR. They achieve 94.62% TPR while the FPR is only 0.6% while using Random Forest.

3.4 Static Analysis Based Approaches

Static analysis based approaches are the most common Android malware detection technique due to their simplicity and comparatively high detection rate. While several works deal with static analysis based approaches, this section contains only works that deal with Permissions or API Calls as this study focuses on identifying significant features from Permissions and API Calls.

Arp et al. [14] proposes a lightweight static feature-based malware detection method called Drebin. In the Drebin, they analyze 8 static features, including Permissions and API Calls, of Android and converted them into vector space. Using 5,560 malware samples and 123,453 benign samples, they evaluate their approach. They achieve a 94% detection rate with the limited false positive rate (FPR) and outperforms existing anti-virus scanners.

Karab et al. [15] introduces MalDozer, a system that uses API Call sequences and deep learning techniques to classify Android malware. According to multiple evaluations using different datasets, they achieve an F1-score of 96%-99% with a false positive rate (FPR) of 0.06%-2%.

Yerima et al. [17] proposes a parallel machine learning-based malware classification approach using API Calls, Permissions, and Commands. They combine rule-based classifiers, function-based classifiers, tree-based classifiers, and probabilistic classifiers to detect malware. They assess their approach using 2925 malware apps and 3938 benign apps. Using 179 features, they gain accuracy of 97.3%, FPR of 3%, and FNR of 2.8% which outperform individual classifiers. This work also surmounts their prior work [16] based on the Bayesian classifier.

Mariconti et al. [27] suggest a behavioral model based on Markov Chain using API Calls call graph. They evaluate their approach using 35,493 malware samples from Drebin and VirusShare and 8,447 benign samples. They achieve F-measure as close to 99% and outperforms one of state-of-the-artwork - DroidAPIMiner [3].

Peiravian et al. [28] combines Permissions and API Calls and shows that the combination performs better than the individual feature. They use 1200 malware apps from Android Malware Genome Project and 1200 benign apps from Play-Store. Using API Calls modeled with SVM (Support Vector Machine) classifier, they gain accuracy of 95.75%, a precision of 91.7%, a recall of 94.8%, and an AUC of 0.957.

Su et al. [29] employ deep learning for feature learning and Android malware detection. They use Permissions, API Calls, Actions, and App Components for their deep learning model - Deep Belief Network. They evaluate 3986 benign apps from PlayStore and 3986 malware apps from Drebin, Android Malware Genome Project, and Contagio Dump. They outperform several existing works with a 99.4% detection rate using the SVM (Support Vector Machine) classifier. Also, it can analyze an app in 6 seconds on average which increases its feasibility.

Raphael et al. [30] uses X-ANOVA, a modified version of conventional ANOVA, to rank features for malware detection. They deal with Permissions, Opcodes, and Methods and employs three classifiers, such as Random Forest, J48, and AdaBoostM1. They achieve 90.63% accuracy using the ranked features.

Manilyzer, introduced by Feldman et al. [31] uses Permissions and other manifest file features for malware detection. They use 307 malware apps from Contagio and 310 benign apps from PlayStore for evaluation. Their approach gains 90% accuracy with the J48 classifier. But their false positive rate (FPR) is 10.1% and the false-negative rate (FNR) is 10% which restricts its applicability.

Dhanya et al. [32] used Permissions and API Calls for Android malware detection. Separability assessment Criteria is used for feature selection in this research. Using the 77 selected features and four different machine learning algorithms (Naive Bayes, SVM, J48 & Random Forest), they evaluated their work. Their performance regarding F-measure, precision, and recall is dubitable as they used Drebin, an outdated and limited dataset. Besides they did not consider any other features except Permissions and API Calls.

3.5 Dynamic Analysis Based Approaches

Dynamic analysis based approaches are less common due to their high complexity and time-consuming nature. They take notably more time than static analysis as they analyze run time behavior of Android application rather than using static features. Moreover, dynamic analysis is the least related type of malware detection technique with respect to this study as no dynamic features are employed here. So, a brief overview of some dynamic analysis based approaches is described in this section.

Amos et al. [33] propose a framework - STREAM for dynamic analysis for Android malware detection. It collects dynamic information on memory, network, battery, binder, and permissions. It uses 1330 malware apps from Android Malware Genome Project, VirusTotal, and 408 benign apps from PlayStore. Using combined features and different classifiers (Naive Bayes, Random Forest, Bayes net, Multilayer perceptron, Logistic, and J48), it gains 68.75% to 81.25% detection accuracy.

Another work, DroidDolphin by Wu et al. [34] uses 13 types of run time activities like network data, file operations, information leaks, SMS, API Calls, etc. for malware classification. They analyze 32,000 benign apps and 32,000 malware apps using Support Vector Machine (SVM). They achieve 86.1% accuracy with an F1-score of 0.857.

Dash et al. [35] present DroidScribe, where it employs dynamic analysis using run-time behaviors like Binder, System Calls, File Operations, Network Access, etc. They evaluate their work using the Drebin dataset of 5560 malware samples. With the Support Vector Machine (SVM) classifier, they improve detection accuracy from 72% to 84%. Finally, they propose a hybrid prediction technique that improves the detection accuracy from 84% to 94%.

Afonso et al. [36] employ API Calls frequency and System Calls frequency for Android malware classification. They use 4552 malware samples from the Android Malware Genome Project and VirusShare. Also, 3831 benign apps from the AndroidPTI market are used. They classify malware using various classifiers like Random Forest, J48, Simple Logistic, Naive Bayes, etc and Random Forest outperforms the rest of the classifier. They achieve a 96.82% detection rate with high precision, recall, and F-measure. Their false-negative rate is 3.94% and the false positive rate is 2.43%.

3.6 Hybrid Analysis Based Approaches

Hybrid analysis based approaches gain researchersáttention due to their high detection rate and all-inclusiveness. Recently, there are many works that deal with hybrid analysis and explore opportunities in this field. Most of the hybrid works mainly focus on static features while combining static and dynamic features. Only works dealing with Permissions or API Calls as static features are included in this section. A brief overview of hybrid analysis based approaches is discussed here.

One of the state-of-the-artwork in hybrid analysis, Marvin [37] employed a lot of static and dynamic features to detect Android malware. It extracted Permissions, Intents, Suspicious Files, API Calls, Developers Certificate, etc. as static features and File Operations, Network Operations, Phone Events, Dynamically Loaded Code, etc. as dynamic features. It used SVM and Linear Classifiers (Regularized Logistic Regression) to build a detection model where Linear Classifiers can detect more accurately but SVM is faster comparatively. For labeled test data, Marvins performance is sound enough as its accuracy to detect malware is 98.24% with less than 0.04% false-positive rate. But for previously unseen malware, its accuracy is close to 90%. Besides, to avoid the obsolescence of its classification model in the future, it presented a retraining strategy. Though Marvin considers a lot of features, it overlooked system-level events such as System Calls: an integral part of the behavioral aspects (dynamic features)

Samadroid [38] presented an on-device malware detection architecture which ensures the resource efficiency by reducing memory overhead of local devices. It used a subset of Drebin's [11] features (6 out of 8) as static features and 10 predefined System Calls as dynamic features. Its accuracy is almost 98% with a false positive rate of 0.1%. Though it incorporated System Call into its feature space and outperform Drebin [11], it used the old dataset. Thereby it might fail to fight against recent malware as malware behavior changes frequently over time. It also overlooked any additional dynamic features.

BRIDEMAID [39] proposes a framework using multi-level and multi-feature analysis. It can detect polymorphic and composition malware to avoid zero-day attacks. Its accuracy is relatively high regarding existing works. However, it does not use any benchmark dataset. Also, it reports only accuracy and FPR, other metrics should be reported to properly evaluate the framework.

OmniDroid [40] fuses several prior tools to extract many static and dynamic features and employs ensemble-based classifiers. Though they considered only a large feature-set, their performance is relatively lower than existing works.

MADAM [41] concurrently assesses static and dynamic features at four levels in detecting mischievous activity. Though it gains accuracy of 96.9%, it has high memory overhead and limited scope (only run in the rooted device, works on post-installed apps).

Hadm [42] incorporated Deep Neural Network for feature extraction from a set of static and dynamic features. It exhibited that combining advanced features derived by deep learning with the original static and dynamic features provides consequential returns. It achieved 94.7% accuracy with a false positive rate of 1.8% while with the original features the best accuracy is 93.5%. An improvement of 1.2% with the cost of complexity.

Droid-detector [43] extracted more than 200 static and dynamic features with the deep neural network. It achieves 96.5% accuracy in detection. However, it uses a limited dataset and limited types of features.

Mobile-SandBox [44] used Permissions, Services, Receivers, Intents, Potentially dangerous functions, and methods as static features and investigated Native Code (Native API Calls) and Network Traffic as dynamic features to classify malware. It lacks in performance as it did not provide any solid performance metrics.

Kapratwar et al. [45] used Permissions and System Calls for hybrid analysis. Its performance (AUC) is significantly better for static features in comparison with dynamic features. But it used a small (200 apps) and old dataset and overlooked other static and dynamic features.

Liu et al. [46] proposed a hybrid malware detecting scheme for Android where Permissions and API Calls are used as static features and System Calls used as dynamic features. Their scheme's detection accuracy is from 93.33according to experimental results. Though they considered only a small feature-set and their dataset are also limited. Table II depicts the literature overview of hybrid analysis using machine learning.

Patel et al. [47] use a genetic algorithm for rule-based malware classification using hybrid features. By assessing more than 231 features, it achieves 96.4% accuracy in malware detection. But it uses a limited dataset and its execution time and resource consumption are high.

Yusof et al. [48] use Permissions, API Calls, and System Calls while achieving sound performance with respect to the accuracy, precision, and recall. However, its model is trained with the malware samples only which would lead to a biased model. Also, FPR is high enough in their work.

3.7 Summary

According to the literature review, the most common malware app data sets are Drebin, Contagio, and the Android Malware Genome Project. Besides, most researchers use the Google Play Store and local app stores to collect benign applications. VirusTotal, VirusShare, etc. sources are also used for malware samples.

Random Forest and the Support Vector Machine (SVM) are the most used machine learning techniques in Android malware detection. Apart from that, Naive Bayes, KNN, J48, Logistic Regression, etc. are also common in the existing research. Researchers use many evaluation metrics in malware detection, such as accuracy, true positive rate (TPR), false-positive rate (FPR), precision, recall, and f1-score.

Permissions and API Calls are the most used static features and System Calls are the most used dynamic features. Existing works suggest that these two features play a vital role and have a notable impact on detecting Android malware accurately. Therefore, researchers give the API Calls and Permissions features the most priority in all types of Android malware detection approaches except dynamic analysis based approaches.

From the discussion of the existing works in Android malware detection techniques, it is evident that the increasing number of Permissions and API Calls hardens the classification by imposing complexity. Only a handful of works deal with this issue of large features space and try to reduce it by identifying important features while maintaining detection performance. However, it is also visible that exploring important features is still challenging. Besides, combining multiple types of features and finding out important features from them also seems to be challenging. However, up-to-date machine learning techniques, feature selection algorithms hold promises to pave the way towards mitigating these issues.

Identifying significant Permissions or API Calls, especially significant ensemble features can help to reduce complexity without compromising detection performances. So, researches that focus on sorting out significant features can solve the aforementioned problems.

Chapter 4

Significant Features Analysis

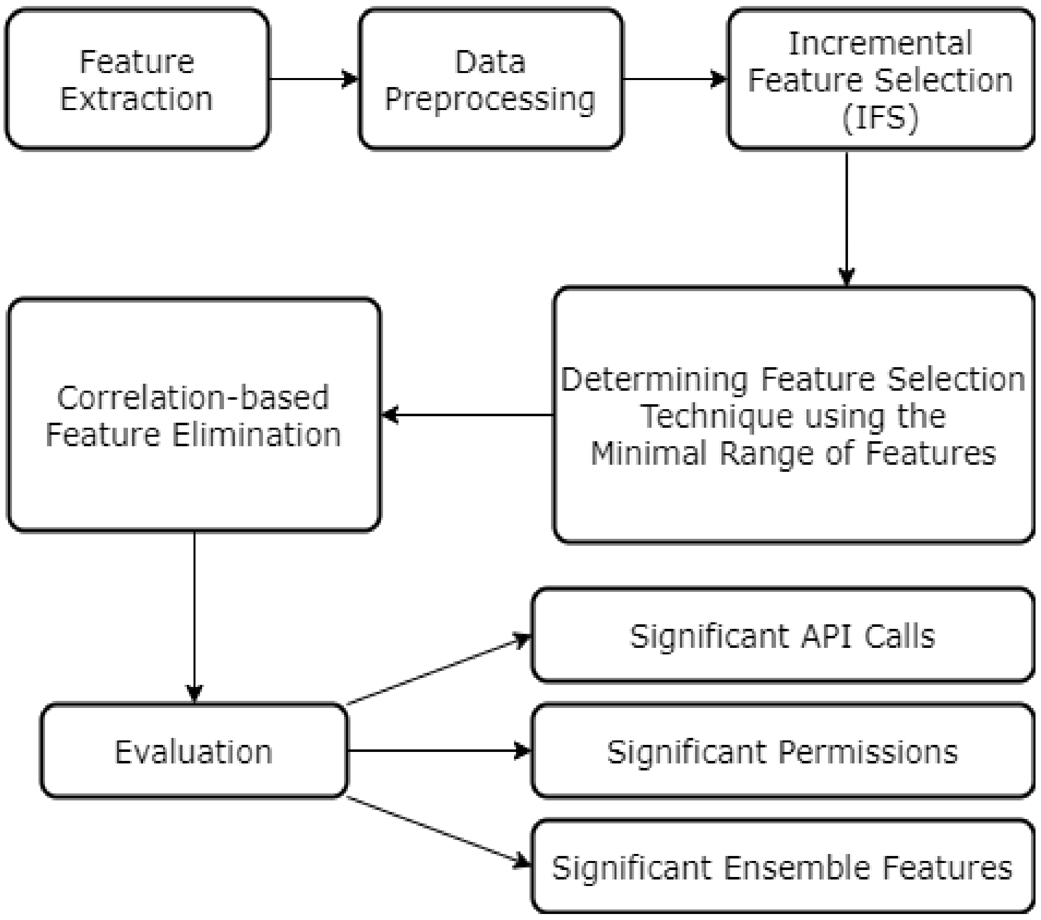
BETTER TO GIVE A TABLE AS A SNAPSHOT OF PREVIOUS WORKS AND HIGHLIGHTING THE DIFFERENCE OF THIS WORK WITH OTHERS.

A lot of work is being conducted on static analysis [14, 15, 16, 17, 28, 29, 30, 31, 32], dynamic analysis [33, 34, 35, 36] and hybrid analysis [37, 38, 39, 41, 42, 43, 45, 46, 47, 48] for Android malware detection. Existing research indicates that typical Android malware detection approaches deal with many features while analyzing malware which increases complexity lysis However, several works [1, 2, 3, 4, 5, 6, 7, 8] deal with analyzing significant features. Those works suggest that identifying significant features can reduce complexity while maintaining performance. Therefore, this study aims to analyze significant features in Android malware detection using machine learning techniques. Two of the most frequently used features in Android malware detection, Permissions, and API Calls, are used in this study.

This study analyzes significant features from three aspects, such as:

1. Significant Permissions Analysis
2. Significant API Calls Analysis
3. Significant Ensemble Features Analysis

These three analyzes are performed separately for two reasons. Firstly, the individual significance of Permissions and API Calls can be assessed if those three analyzes are conducted separately. Also, individual significance can be compared



Feature Extraction
Data Preprocessing
Incremental Feature Selection (IFS)
Determining Feature Selection Technique using the Minimal Range of Features
Correlation-based Feature Elimination
Evaluation
Significant API Calls
Significant Permissions
Significant Ensemble Features

Figure 4.1: Overview of Significant Features Analysis

with similar research. There are several distinct works which deal with significant Permissions [6, 7, 8], significant API Calls [3, 4, 5], and significant ensemble features [1, 2] independently. Secondly, the sources and extraction processes of Permissions and API Calls are different. Permissions can be extracted more easily than API Calls from *AndroidManifest.xml* file. So, in terms of the applicability of malware detection, apart from detection performance, it should also be considered that which features are easily extracted and analyzed. Thereby, the overall approach is assessed in three distinct aspects.

The overall approach is depicted in Figure (4.1). First, the features from Android application files are extracted and converted into features vector. Permissions and API Calls are extracted separately. Then, basic data preprocessing is performed on the extracted features. Afterward, the cornerstone of this approach, incremental feature selection (IFS) is carried out. Several feature selection techniques are assessed incrementally to avoid any predefined biases, like selecting top k features. From the IFS, a minimal range of features for different feature selection techniques are identified using performance plots. Next, the optimum number of features and best-suited feature selection techniques for malware detection are identified by analyzing the performance. After identifying the optimal/minimal number of features, a correlation-based feature elimination strategy is performed for further reduction of features set. Finally, the reduced set of features are evaluated on the three above mentioned aspects. That evaluation is performed based on accuracy, precision, recall, f1 score, ROC-AUC, execution time, and comparison with existing works.

4.1 Feature Extraction

In order to detect Android malware, static features like Permissions and API Calls are the most frequently used [1, 2, 3, 4, 5, 6, 7, 8, 14, 15, 27, 28, 29, 30, 31, 32] features. Even in hybrid analysis of Android, these two features are quite common [37, 38, 39, 41, 42, 43, 45, 46, 47, 48]. Permissions and API Calls are extracted from the Android application files separately. In doing so, the application file is to be uncompressed first using any reverse engineering technique. There are several reverse engineering tools available for doing that, such as APKTool, APKInspector, Androguard, etc. After reverse engineering, the *AndroidManifest.xml* file and the smali files are needed for extracting Permissions and API Calls respectively.

AndroPyTool [49] is used in this study for feature extraction. It generates JSON files containing Permissions and API Calls while performing static analysis of the Android app. Afterward, the JSON files are extracted for mining Permissions and API Calls. Finally, the extracted features are converted into feature vectors through parsing. The feature vectors are used for data analysis and clas-

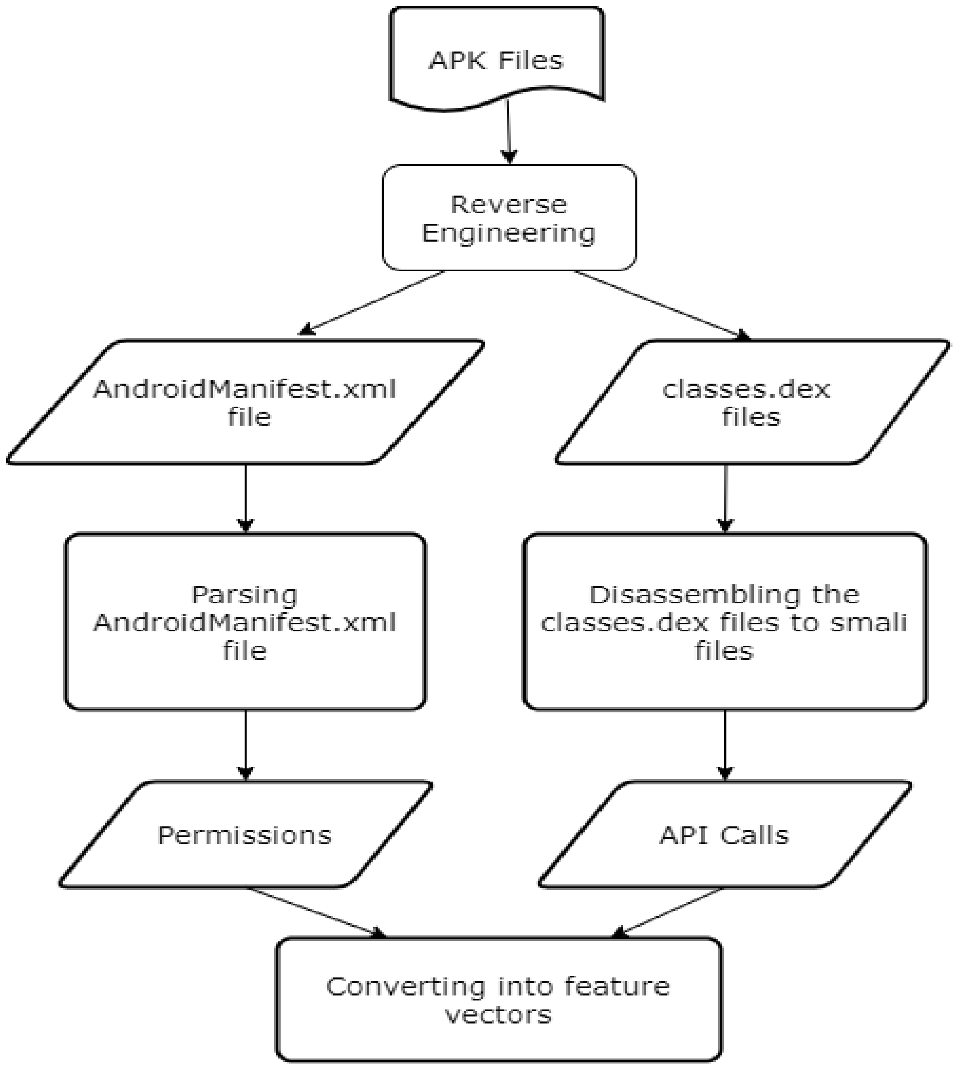

APK Files
Reverse Engineering
AndroidManifest.xml file
classes.dex files
Parsing AndroidManifest.xml file
Disassembling the classes.dex files to smali files
Permissions
API Calls
Converting into feature vectors

Figure 4.2: Feature Extraction of Android App

sifying Android malware. Overall feature extraction process is depicted in Figure (4.2)

4.1.1 Permissions Extraction

Android provides a scheme of install-time permissions system for managing access to related APIs for privacy and security. For example, android.permission.internet is the permission to open the socket for internet connection required for an application. The permissions are requested upon installing an app and outlined in the *AndroidManifest.xml* file explicitly. The manifest file is being parsed into *.xml* format using the Androguard tool to extract the Permissions of an app.

4.1.2 API Calls Extraction

Android apps are developed in Java or Kotlin and compiled into optimized bytecode for the Dalvik virtual machine (DVM). The *classes.dex* files contain the bytecodes. API Calls can be obtained by disassembling the *classes.dex* files into *smali* files. Here, *baksmali* is used as the disassembler.

4.2 Data Preprocessing

The dataset is preprocessed using traditional data preprocessing techniques. Missing value treatment is and label encoding is incorporated. The data type of all features in the features space is converted into a singular format for coherence. For, incremental feature selection, the dataset is split into training and validation sets in a ratio of 0.5.

4.3 Incremental Feature Selection (IFS)

In identifying the significant features for Android malware detection, two aspects are considered. The incremental feature selection approach plays a crucial role in both aspects to conclude.

Firstly, how many numbers of features should be selected for identifying significant features? As the goal is to identify the significant features, the first concern is about the number of features to be selected. It is possible to adjust the number of features using some predefined threshold or parameters for the feature selection algorithms. For example, the ROC-AUC score or the Mutual Information Gain can be used to limit the number of features beforehand.

In this study, it is avoided to set any predefined parameters or threshold for selecting the number of features in advance. Rather, it is intended to determine the optimal/minimal number of features by analyzing performance metrics for different numbers of features. In doing so, several feature selection techniques are analyzed incrementally. For each feature selection technique, from one to the highest number of features are assessed separately based on the performance metrics (accuracy, precision, recall, f-1 score, and AUC). The performance metrics are visualized in plots to understand the increasing pattern of performances according to the number of features.

Secondly, which feature selection technique is better suited to reducing the number of features while preserving performances in Android malware detection? Again, several feature selection techniques are analyzed incrementally to figure out the most suitable technique, rather than imposing a predetermined feature selection technique. The feature selection techniques used in this study are briefly discussed in the following subsections.

4.3.1 IFS using Mutual Information Gain (Entropy-Based)

Mutual Information is a non-negative value between two random variables, which measures dependency between variables. It measures the quantity of information gained by analyzing the other random variable involving one random variable. It is equal to zero if there are two independent random variables, and higher values mean higher dependence. The function is based on non-parametric methods based on entropy estimation of the distances from k-nearest neighbors [50, 51]. The mutual information of two jointly discrete random variables X and Y is calculated as a double sum using Equation (4.1).

$$I(X;Y) = \sum\sum p(X,Y)^{(x,y)\log\left(\frac{p(X,Y)^{(x,y)}}{p(X)^{(x)}p(Y)^{(y)}}\right)} \tag{4.1}$$

where,

p(X,Y) is the joint probability mass function of X and Y,

p(X) and p(Y) are the marginal probability mass functions of X and Y respectively.

4.3.2 IFS using Univariate ROC-AUC Score

A ROC curve (receiver operating characteristic curve) is a graph representing a classification model output at all classification thresholds. This curve maps two parameters: True Positive Rate and False Positive Rate. AUC stands for "Area under the ROC Curve," meaning that AUC measures the whole two-dimensional space under the ROC Curve. The region under the curve (AUC) is proportional to the probability that a classifier ranks a randomly selected positive instance higher than a randomly selected negative one by using normalized units [52]. Univariate ROC-AUC involves the analysis of a single variable. An AUC equal to 0.5 corresponds to a type of random classification. For a model to be acceptable AUC will be greater than 0.5.

4.3.3 IFS using Recursive Feature Elimination (RFE)

The goal of recursive feature elimination (RFE) is to pick features by recursively considering smaller and smaller sets of features, given an external estimator that assigns weights to features. First, the estimator is trained on the initial collection of features and the importance of each function is obtained. The least significant characteristics are then pruned from the present range of characteristics. The process is repeated recursively on the pruned collection before finally achieving the required number of features to be chosen [53]. In this work, two classifiers are used as the estimators of the RFE.

RFE with Gradient Boosting Classifier

As the base estimator of the RFE, Gradient Boosting Classifier is employed. Gradient Boosting Classifier builds an additive model in forward-stage-wise fashion; enables arbitrary differentiable loss functions to be optimized. Regression trees are fit on the negative gradient of the function of binomial or multinomial loss of deviance in each point [54].

RFE with Random Forest Classifier

Random Forest The classifier is also used as the base estimator of the RFE. It is a meta-estimator that fits multiple decision tree classifiers on various dataset sub-samples and uses an average to improve predictive [55].

4.3.4 IFS using SelectKBest with chi2

SelectKBest scores the features according to the k highest scores. It takes a score function as a parameter, which would be specific to a pair. The score function retains the features of the first k with the highest scores [56]. In this study, the chi2 scoring function is employed. This scoring function computes the chi-squared stats between each non-negative feature and class scores accordingly. It tests for which the distribution of the test statistic approaches the χ^2 (Chi-Squared) distribution asymptotically [57].

4.3.5 IFS using SelectFromModel(Tree-Based)

SelectFromModel is a meta-transformer that can be used along with any tree-based estimator. It calculates the feature importance of each feature according to fitting the estimator into the data. Based on the feature importance it selects the top N features, where N is predefined [58]. Tree-based estimators are used here as it can classify the significant features by selecting the classification features based on how well they boost the node's purity [59]. In this case, every possible value of N is evaluated. Also, two tree-based estimators are incorporated here: Random Forest Classifier and Extra Trees Classifier.

4.4 Determining Feature Selection Technique using the Minimal Range of Features

After carrying out the incremental feature selection using different feature selection techniques, analysis of performance metrics is evaluated to identify the minimal range of features. The minimal range of features implies a range of features from which segment the performances of Android malware detection are not increased notably with respect to the increasing number of features. In other words, before the minimal range, the performances are increased. But, after the minimal range, the performances are quite unchanged with the increase of features. To conclude the analysis, a self-explanatory plot is generated using the performance metrics (accuracy, precision, recall, f-1 score, and AUC) with respect to the increasing number of features. According to the plots for different techniques, the minimal range of features are deduced.

An example of analyzing the minimal range of features is depicted in Figure (4.3). The minimal range of features is marked with a black rectangle in the figure. According to the figure, as the number of features increases, the performances are also increasing notably. However, from the minimal range of features, the performances are getting stable with respect to the increasing number of features. After the minimal range, the lines remain almost horizontal with respect to the number of features. This indicates that in the minimal range, the performances of malware detection reach as close to the final performances. So, in order to reduce complexity while keeping performance as close to the final performance, the minimal number of features is selected to analyze significant features in Android

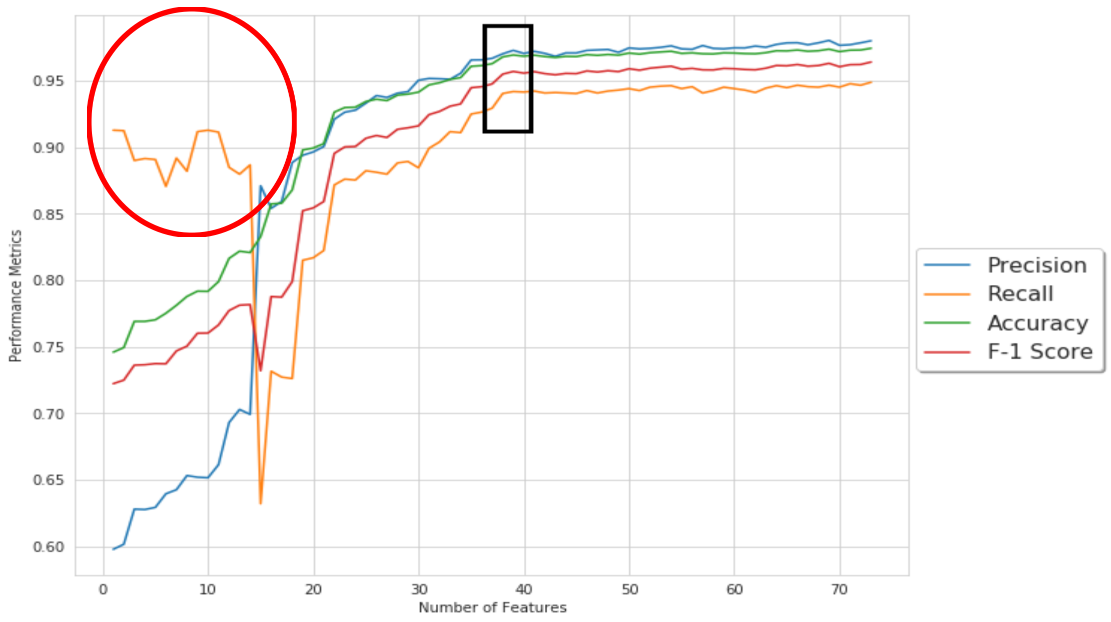


Figure 4.3: Identifying Minimal Range of Features

malware detection.

The minimal ranges for different feature selection techniques are conducive to determine the suitable feature selection technique. The feature selection technique which provides the lowest minimal range is carefully chosen for identifying significant features in Android Malware Detection. As the minimal range of features offers almost similar performances to the full features set, taking the lowest minimal range of features from several feature selection techniques is advantageous regarding the reduction of complexity.

4.5 Correlation-based Feature Elimination

In the final step, a feature elimination strategy is performed for further reduction of features without affecting the performances considerably. This step is applied to the important features obtained from the selected feature selection technique. Here, a correlation-based feature elimination strategy is applied. The strategic intuition is that if two features are highly correlated with regards to malware detection, removing one of them might not affect the overall performances.

A pair-wise Pearson correlation coefficient is calculated for all pairs of important features [60]. It is a measure of the linear correlation between two variables X and Y. It is calculated using the following Equation (4.2).

$$\rho_{xy} = \frac{Cov(x, y)}{\sigma_x, \sigma_y} \tag{4.2}$$

where,

ρ_{xy} = Pearson correlation coefficient

Cov (x,y) = covariance of variable x and y

σ_x = standard deviation of x

σ_y = standard deviation of y

Afterward, all the pairs with a Pearson correlation coefficient less than 0.85 are filtered out for the feature elimination process. As the two features in each pair are highly correlated, so removing one of them would not affect the classification performances.

The next concern is which feature to eliminate from a pair and how to choose that feature. The elimination strategy here is to remove the less important feature from each pair using their relative feature importance. To measure the relative importance of the features, a tree-based estimator – Random Forest Classifier is used. According to the relative feature importance, the more important feature in each pair is intact, and the less important feature is eliminated.

4.6 Evaluation Strategy

Finally, the reduced set of features are evaluated on the accuracy, precision, recall, f1 score, ROC-AUC, execution time, and comparison with existing related works.

The evaluation is carried out separately on 3 aspects as follows:

1. Significant API Calls Analysis
2. Significant Permissions Analysis
3. Significant Ensemble Features Analysis

Five performance metrics are considered for the evaluation – accuracy, precision, recall, f1 score, AUC. These five performance metrics are selected based on their relevance in this analysis, and taking into account the evaluation's comprehensiveness. Existing studies often use these performance metrics but not all of the five metrics are included in a single analysis. A brief description of the performance metrics and their relevance is discussed as follows.

- **Accuracy:** Accuracy is the most intuitive performance metric. Accuracy denotes the ratio of correctly classified samples with respect to all samples [61]. Accuracy is calculated using the following Equation (4.3).

$$Accuracy = \frac{True\,Positive + True\,Negative}{Total\,Population} \tag{4.3}$$

  Accuracy is the most frequently used performance metric in Android malware detection. It signifies the ratio of correctly classified malware to all samples.

- **Precision:** Precision denotes the ratio of correctly classified samples with respect to all classified samples [61]. Precision is calculated using the following Equation (4.4).

$$Precision = \frac{True\,Positive}{True\,Positive + False\,Positive} = \frac{True\,Positive}{Total\,Classified\,Positive} \tag{4.4}$$

Precision is useful where False Positive costs are high. A false positive in malware detection implies the classification of a benign app as malware. Therefore, if the precision is lower, users will not use many benign apps due to the classification of many benign apps as malware.

- **Recall:** Recall denotes the ratio of correctly classified samples with respect to all actual samples [61]. Recall is calculated using the following Equation (4.5).

$$Recall = \frac{True\,Positive}{True\,Positive + False\,Negative} = \frac{True\,Positive}{Total\,Actual\,Positive} \tag{4.5}$$

  Recall is useful where False Negative costs are high. A false negative in malware detection implies the classification of a malware app as benign. Therefore, if the recall is lower, users will suffer the most by running malware as a benign app.

- **F1 Score:** F1 score is a function of precision and recall which balances between them. Hence, it takes both false positives and false negatives into account [61]. F1 score is calculated using the following Equation (4.6).

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.6}$$

  F1 score is useful in malware detection because both False Positive and False Negative costs are high here. Because of misclassification users neither want to use malicious app nor want to skip benign apps.

- **AUC (Area Under Curve):** ROC (Receiver operating characteristic) is a probability curve and AUC denotes the degree or measure of separability. It shows how well a model can differentiate between groups [62].

  In malware detection, the higher the AUC, the better the model is at distinguishing between benign and malware apps. It is a good measure to justify the model's applicability in malware detection.

Here, the performance metrics play a vital role in determining the minimal range of features for different feature selection techniques. Besides, the best-suited feature selection technique is chosen using the plots derived from these performance metrics. The selected feature selection technique is assessed eventually within the minimal range of features for the evaluation.

In the final assessment, the Random Forest classifier is trained and evaluated with 10-fold cross-validation. The dataset is split equally for feature selection and evaluation respectively. 10-fold cross-validation is employed in both parts.

Along with the five performance metrics, execution time, and comparison with the existing approaches are also evaluated. To minimize internal and external biases the execution time is taken as the mean of running the model 5 times. Also, all the models are executed in the same machine. While comparing with the existing approach, the data sets, performance metrics, selected features (Permissions and API Calls) are considered. Existing works that do not use Permissions or API Calls, for example, are excluded from the comparison. Similarly, the most common performance metrics, such as accuracy is used in the comparison.

Finally, significant features are identified using the relative feature importance and reported accordingly. All these evaluation strategies are performed on the above mentioned three aspects separately.

4.7 Summary

This chapter includes a detailed description of the overall approach. Firstly, feature extraction and data-preprocessing are performed using AndroPyTool and Weka tools. Then, incremental feature selection (IFS) techniques are carried out for several feature selection algorithms and their corresponding performance is plotted for analysis. According to that, the best-suited technique and corresponding minimal number of significant features are identified. Also, a correlation-based feature elimination step is performed to reduce the number of features without hampering performance. Lastly, an evaluation strategy is defined for analyzing significant features in Android malware detection.

Chapter 5

Evaluation

This chapter contains the dataset description, implementation details, experimental setup, and evaluation of significant features analysis. The discussion of the results to explain some important insights are also presented.

The feature extraction, data preprocessing, incremental feature selection (IFS), performance plotting, correlation-based feature elimination steps are implemented in Python using Jupyter-Notebook and PyCharm platforms.

where is weka tool used?

In the evaluation of this study, two benchmark datasets are used. The experimental results are analyzed based on five performance metrics - accuracy, precision, recall, f-1 score, and AUC. These metrics are widely used in the performance measure of Android malware detection. Besides, the execution time of detection and comparison with existing works is assessed.

5.1 Implementation

In this section, the tools and technologies for implementing the proposed technique are discussed. The class diagram of the implementation is also provided to get an overall view of the technique.

5.1.1 Tools and Technologies

The tools and technologies to implement the overall technique are described in this subsection. Firstly, Omnidroid is used for feature extraction. Python is used for creating feature vectors from the extracted features (JSON format). In the data preprocessing and data-exploration, the Weka tool is used. Finally, incremental feature selection, identifying minimal ranges, plotting performances, correlation-based feature elimination, model building, and performance evaluation are implemented using Python in Jupyter Notebook computational environment.

The following tools and libraries were used to develop the system.

- AndroPyTool [63]: AndroPyTool is used to extract static and dynamic features from Android applications. In this study, only static features are extracted using this tool. It integrates numerous well-known analytics tools for Android applications, such as DroidBox, FlowDroid, Strace, AndroGuard, and VirusTotal. It analyzes the application and performs reverse engineering to get source files from the apk file. Then, using various integrated tools, it extracts Permissions, API Calls, and other static features and generates feature vectors in JSON and CSV format. These feature vectors are the primary resources for data preprocessing and other steps.

- Weka 3.8 [64]: Weka is a data analysis platform that facilitates the preprocessing and data analysis. Weka offers API to access various machine learning algorithms such as Random Forest, Decision Tree, Support Vector Machine (SVM), etc. In this study, Weka is primarily used for data preprocessing.

- Jupyter-Notebook [65]: The Jupyter Notebook is an open-source software application that enables one to create and exchange live data, calculations, visualizations, and narrative text documents.

The implementation was carried out on an operating system with following specifications.

- Operating System: Windows 10
- RAM: 16.00 GB
- CPU: 3.70GHz AMD Ryzen 5 3400G with Radeon Vega Graphics
- Platform: 64 bit

5.2 Datasets

In this study, the Drebin [14] and the Android Malware Genome Project [66] datasets are used. According to the literature review, these two datasets are the most frequently used datasets in Android malware detection. Hence it is convenient to use such datasets to compare this study with existing works. The datasets are used separately, instead of being compiled into one to ensure the approachś applicability and generalizability. The description of the datasets is depicted in Table 5.1.

The Drebin dataset contains 5,560 malware applications from 179 different malware families. Also, 9470 benign applications derived from the Google Play Store are incorporated here for classifying the malware properly.

The Android Malware Genome Project dataset contains 1,200 malware samples that cover most existing Android malware families. Here, 2539 benign applications derived from the Google Play Store are incorporated. In the rest of the study, this dataset is referred to as Malgenome.

Table 5.1: Dataset

Dataset	Samples	Benign Samples	Malware Samples
Drebin	15030	9470	5560
Android Malware Genome Project	3739	1200	2539

Only Permissions and API Calls are considered in this work. In total, 73 API Calls are found in the Drebin dataset and 69 API Calls are found in the Android Malware Genome Project dataset. Similarly, 114 Permissions are found in the Drebin dataset and 113 Permissions are found in the Android Malware Genome Project dataset.

5.3 Significant API Calls Analysis

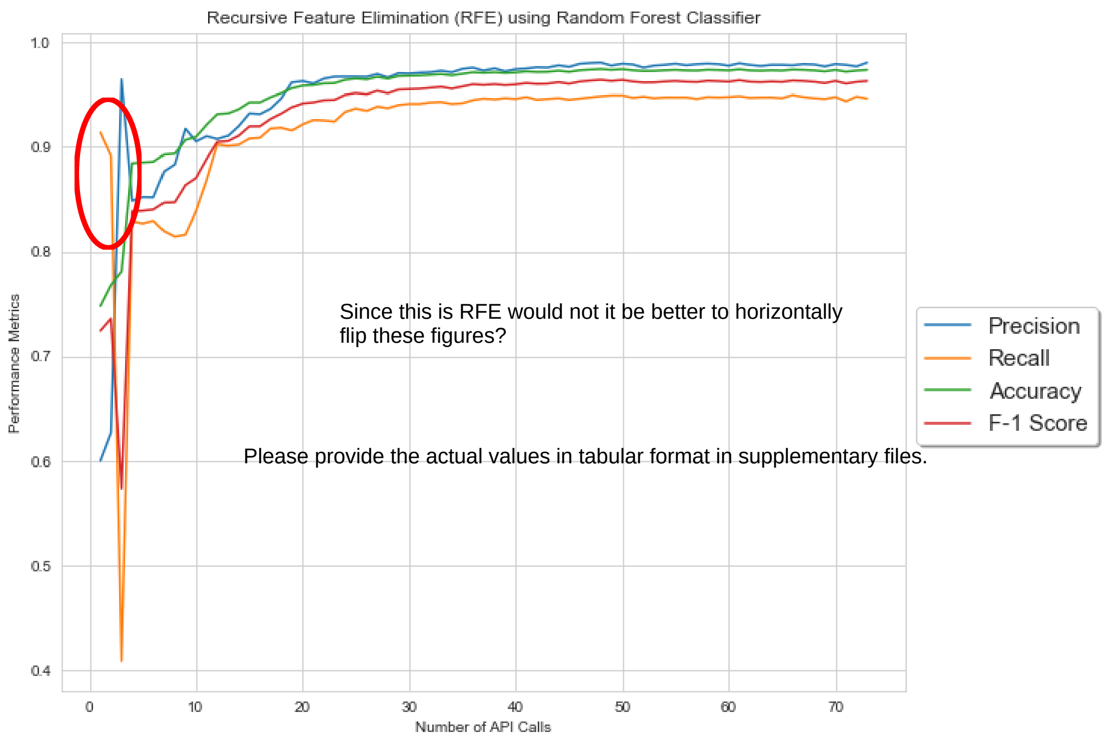
In assessing the significant API Calls in Android malware detection, the proposed workflow is carried out. Only API Calls are evaluated in this subsection to understand the discrete significance of API Calls in Android malware detection. The outcomes and insights from each step of the approach are described in this subsection in order.

5.3.1 Identification of Feature Selection Technique and Minimal Range of Features for API Calls

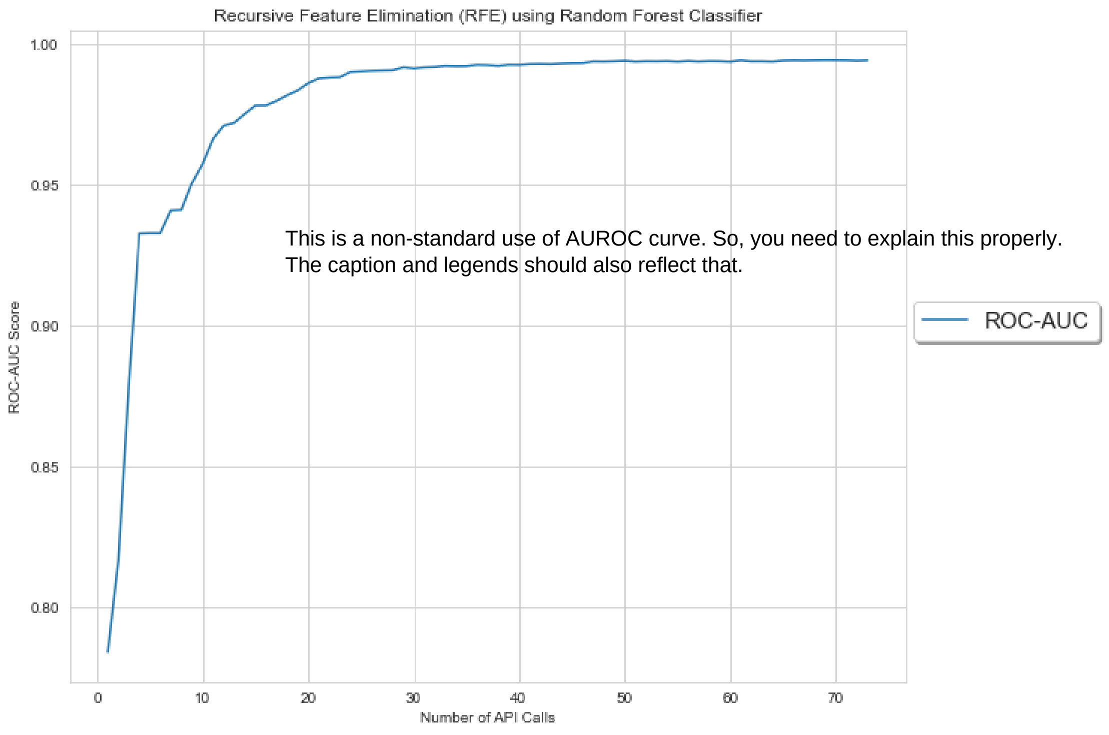
In order to identify the appropriate feature selection technique for malware detection and corresponding minimal range of API Calls, incremental feature selection techniques are performed for all of the feature selection techniques mentioned in Section 4.4. Plotting the performance metrics with regards to the increasing number of API Calls assists in identifying the appropriate feature selection technique and minimal range of API Calls.

Recursive Feature Elimination (RFE) with the Random Forest classifier provides the minimal range of features for both the Drebin and Malgenome data sets. This can be deduced from Fig (5.1) and (5.2) which represents the Drebin and the Malgenome data sets respectively.

For the Drebin dataset, according to the Fig (5.1), as the number of API Calls is increased, the performances (Accuracy, Precision, Recall, F1 score, AUC) ~~are~~

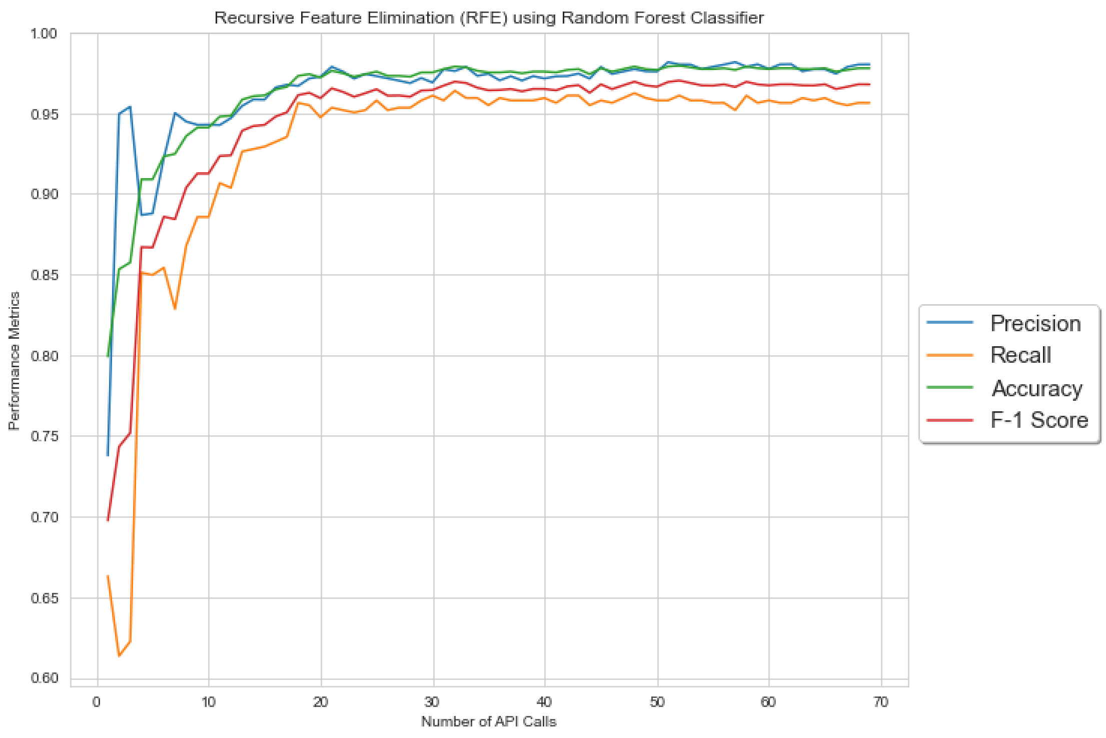


(a) Precision, Recall, Accuracy, F-1 score

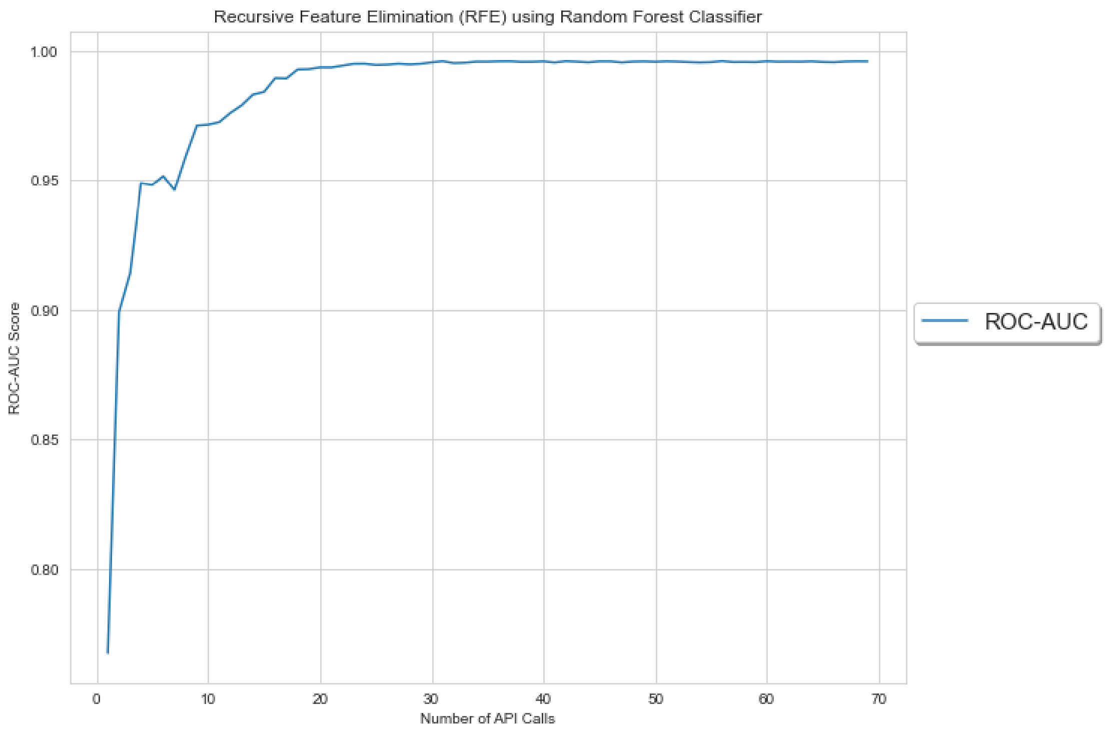


(b) AUC

Figure 5.1: Minimal Range for RFE with Random Forest Classifier for Drebin



(a) Precision, Recall, Accuracy, F-1 score



(b) AUC

Figure 5.2: Minimal Range for RFE with Random Forest Classifier for Malgenome

also increased initially. However, in the range between 20-25 API Calls (approximately), the performance metrics are getting stable and remain constant for the rest of the API Calls. The performance lines remain horizontal with respect to the number of API Calls from the minimal range of features. So, it can be inferred that by taking those 20-25 API Calls, the performances are close enough to the actual performances of all the 73 API Calls.

Similar phenomena can be deduced from Figure (5.2). In the case of the Malgenome dataset, from Figure (5.2), it can be inferred that the minimal range of features is 17-22 API Calls approximately. After that range, the performances remain stable practically. Malware detection performances within the range are similar to taking all of the 69 API calls.

Table 5.2: Minimal Range of Features (API Calls) using Different Feature Selection Techniques

Feature Selection Technique	**Minimal Range for Drebin**	**Minimal Range for Malgenome**
Mutual Information Gain	36-40	43-48
Univariate ROC-AUC Score	35-38	25-30
RFE with Gradient Boosting Classifier	24-30	23-30
RFE with Random Forest Classifier	**20-25**	**17-22**
SelectKBest with chi2	42-47	37-42
SelectFromModel with Random Forest Classifier	23-30	17-25
SelectFromModel with Extra Trees Classifier	25-32	19-24

I am confused on how the non-RFE methods have been implemented. Need more explanation.

The minimal ranges for all the feature selection techniques are depicted in Table 5.2. From the experiments, the best feature selection technique for API Calls reduction is Recursive Feature Elimination (RFE) with Random Forest Classifier as it provides the lowest minimal range among other techniques for both data sets.

RFE with Gradient Boosting Classifier, SelectFromModel with Random Forest Classifier, and Extra Trees Classifier also give a comparatively lower minimal range for both data sets. Other feature selection techniques do not perform equally for both data sets. For instance, the Univariate ROC-AUC Score performs better for the Malgenome dataset but comparatively results in poor for the Drebin dataset.

5.3.2 Correlation-based Feature Elimination (CFE) for API Calls

why was it not performed before?

Before proceeding to the performance evaluation, the correlation-based feature elimination (CFE) is performed on the selected minimal range of API Calls. Experimental results show that CFE can reduce the API Calls without limiting the performances. The reduction of API Calls is depicted in Table 5.3. For instance, CFE can reduce 20 features to 17 and 17 features to 15 features respectively for the Drebin and Malgenome data sets.

Table 5.3: Correlation-based Feature Elimination (CFE) on Minimal Feature Sets for API Calls

Drebin		**Malgenome**	
Number of API Calls in Minimal Feature Sets	**Reduced Number of API Calls with CFE**	**Number of API Calls in Minimal Feature Sets**	**Reduced Number of API Calls with CFE**
20	17	17	15
21	18	18	16
22	19	19	16
23	20	20	19
24	21	21	20
25	21	22	21

5.3.3 Performance of Significant API Calls in Malware Detection

Finally, the reduced set of Significant API Calls are evaluated based on the performance metrics, execution time, and comparison with existing works.

Performance Metrics of the Significant API Calls

The performance evaluation for the significant API Calls are based on five metrics. The evaluation is described in Table 5.4 and 5.5 for the minimal range of API Calls. Table 5.4 shows that for the Drebin dataset, the performance metrics using significant API Calls are close to the performance metrics of using all the API Calls (73). Table 5.5 also shows that for the Malgenome dataset, the significant API Calls performs almost identically to the the full feature set of API Calls (69). Specifically, how many significant API calls should be selected? - It depends on the requirement of the stakeholders. However, it is suggested to use the range of 17-21 significant API Calls based on the performances.

So one of your comments/ hypothesis earlier that: "These growing numbers of features make it complex and would misguide classifiers by over-fitting of data." is not valid in this case.

Table 5.4: Performance Evaluation of Significant API Calls for Drebin

Feature Selection Techniques	Number of API Calls	Reduced API Calls with CFE	Accuracy (%)	Precision (%)	Recall (%)	F1	AUC
None (All Features)	73	-	97.38	98.07	94.63	0.9631	0.9941
RFE with Random Forest Classifier	20	17	95.60	95.76	91.95	0.9379	0.9846
	21	18	95.77	95.71	92.50	0.9405	0.9869
	22	19	95.98	96.33	92.46	0.9433	0.9881
	23	20	96.08	96.51	92.54	0.9446	0.9877
	24	21	96.46	96.86	93.27	0.9501	0.9895
	25	21	96.50	96.94	93.31	0.9507	0.9896

Among other things, to me it would be interesting to compare esults of the set-21 directly found without CFE vs. set-21 found after CFE.

Table 5.5: Performance Evaluation of Significant API Calls for Malgenome

Feature Selection Techniques	Number of API Calls	Reduced API Calls with CFE	Accuracy (%)	Precision (%)	Recall (%)	F1	AUC
None (All Features)	69	-	97.79	98.03	95.64	0.9678	0.9960
RFE with Random Forest Classifier	17	15	96.74	97.35	93.23	0.9517	0.9898
	18	16	97.26	96.68	95.49	0.9603	0.9921
	19	16	97.26	97.12	95.04	0.9600	0.9935
	20	19	97.63	97.73	95.49	0.9654	0.9932
	21	20	97.47	97.43	95.34	0.9632	0.9942
	22	21	97.53	97.72	95.19	0.9639	0.9942

Execution Time of the Significant API Calls

Table 5.6 shows the comparative execution time of malware detection. The result shows that using the significant API Calls (ranges between 17-21 for Drebin and

15-21 for Malgenome), the execution time of the malware detection is considerably lower than using all the features. For large data sets, this time would be substantially higher.

Table 5.6: Execution Time of Significant API Calls

Drebin		Malgenome	
Number of API Calls	**Execution Time (s)**	**Number of API Calls**	**Execution Time (s)**
73 (All)	7.13	69 (All)	6.98
21	4.15	21	4.29
20	4.11	20	4.29
19	4.04	19	4.10
18	3.99	16	4.01
17	3.91	15	3.95

Comparison with Related Works (Significant API Calls)

Significant API Calls are also evaluated in comparison with several related works that deal with significant/important API Calls. Table 5.7 shows the comparative analysis of the performance metrics. This studyś significant API Calls analysis is regarded as SigAPI (21) in the table. Using only 21 significant API Calls for the Drebin dataset, SigAPI (21) performs better than other related works except for the accuracy of DroidAPIMiner [3]. Though DroidAPIMiner uses 169 API Calls where SigAPI (21) uses only 21 API Calls.

Table 5.7: Comparison with Related Works (Significant API Calls)

Research	**Number of API Calls Considered**	**Accuracy**	**Precision**	**Recall**	**F1**
SigAPI(21)	21	96.50%	**96.94%**	**93.31%**	**0.951**
DroidAPIMiner [3]	169	**99%**	N/A	N/A	N/A
Zhao et al. [4]	20	93%	93%	N/A	N/A
Alazab et al. [5]	1326	N/A	N/A	N/A	0.943

5.3.4 Top API Calls in Android Malware Detection

Finally, the top 25 API Calls are analyzed and reported. Table 5.8 shows the top 25 significant API Calls in Malware Detection for the Drebin dataset. These API Calls are derived from the feature selection technique – RFE with Random Forest Classifier. Also, these 25 API Calls are almost identical to the Malgenome dataset except 3 API Calls. More data instances would be conducive to generating a generalizable list of significant API Calls. Yet, as this study primarily suggests an approach for significant API Calls, the dataset to dataset it may slightly vary due to the inconsistency and time of data sets.

Table 5.8: Top API Calls in Android Malware Detection

TOP 25 SIGNIFICANT API CALLS	
transact	TelephonyManager.getNetworkOperator
onServiceConnected	Landroid.content.Context.registerReceiver
bindService	Ljava.lang.Class.getField
attachInterface	android.content.pm.PackageInfo
ServiceConnection	TelephonyManager.getLine1Number
android.os.Binder	Ljava.lang.Class.getMethod
Ljava.lang.Class.getCanonicalName	android.telephony.gsm.SmsManager
Ljava.lang.Class.getMethods	TelephonyManager.getSubscriberId
Ljava.lang.Class.cast	Ljava.lang.Object.getClass
Ljava.net.URLDecoder	TelephonyManager.getDeviceId
android.content.pm.Signature	HttpUriRequest
android.telephony.SmsManager	Runtime.exec
ClassLoader	

5.4 Significant Permissions Analysis

In assessing the significant Permissions in Android malware detection, the proposed workflow is carried out. Only Permissions are evaluated in this subsection to understand the discrete significance of Permissions in Android malware detection. The outcomes and insights from each step of the approach are described in this subsection in order.

5.4.1 Identification of Feature Selection Technique and Minimal Range of Features for Permissions

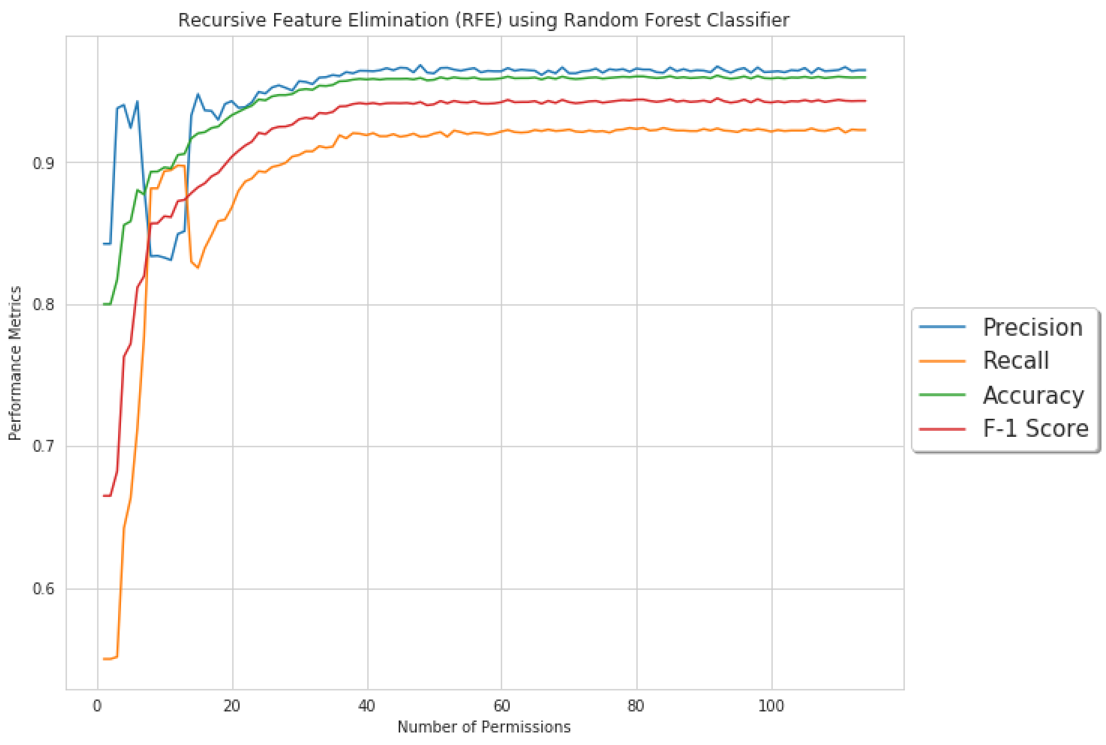
In order to identify the appropriate feature selection technique for malware detection and corresponding minimal range of Permissions, incremental feature selection techniques are performed for all of the feature selection techniques mentioned in Section 4.4. Plotting the performance metrics with regards to the increasing number of Permissions assists in identifying the appropriate feature selection technique and minimal range of Permissions.

Recursive Feature Elimination (RFE) with the Random Forest classifier provides the minimal range of features for both the Drebin and Malgenome data sets. This can be deduced from Fig (5.3) and (5.4) which represents the Drebin and the Malgenome data sets respectively.

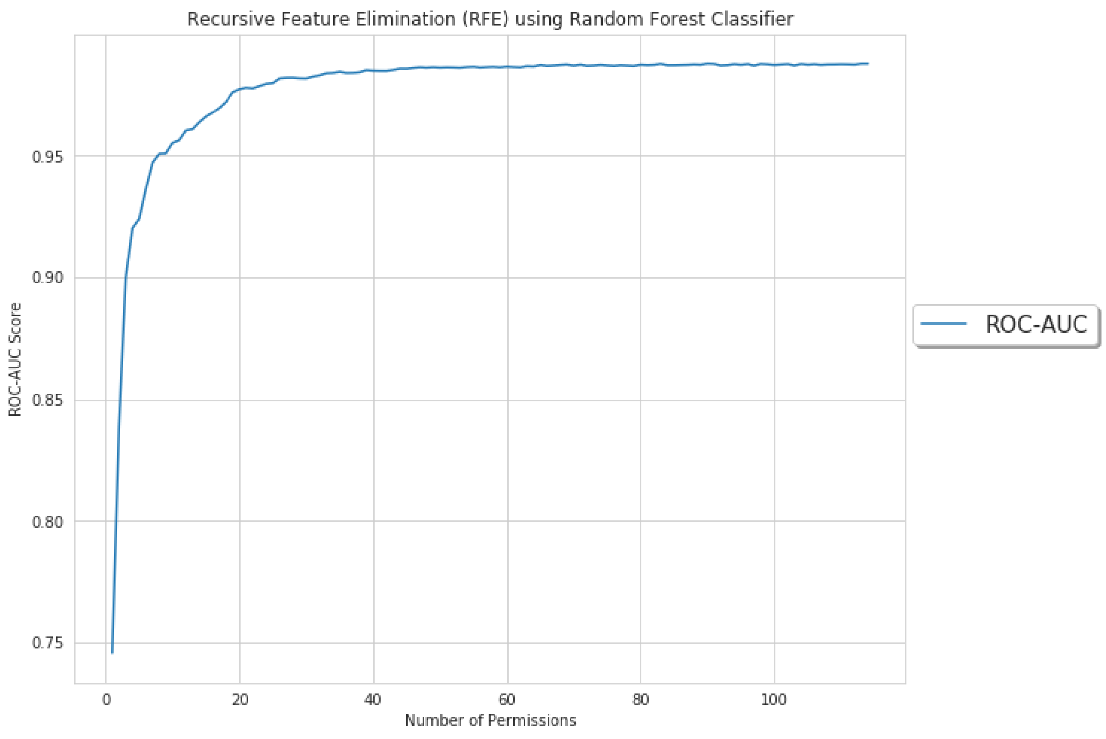
For the Drebin dataset, according to the Fig (5.3), as the number of Permissions is increased, the performances (Accuracy, Precision, Recall, F1 score, AUC) are also increased initially. However, in the range between 34-38 Permissions (approximately), the performance metrics are getting stable and remain constant for the rest of the Permissions. The performance lines remain horizontal with respect to the number of Permissions from the minimal range of features. So, it can be inferred that by taking those 34-38 Permissions, the performances are close enough to the actual performances of all the 114 Permissions.

Similar phenomena can be deduced from Figure (5.4). In the case of the Malgenome dataset, from Figure (5.4), it can be inferred that the minimal range of features is 23-27 Permissions approximately. After that range, the performances remain stable practically. Malware detection performances within the range are similar to taking all of the 113 Permissions.

The minimal ranges for all the feature selection techniques are depicted in Table 5.9. From the experiments, the best feature selection technique for Permissions

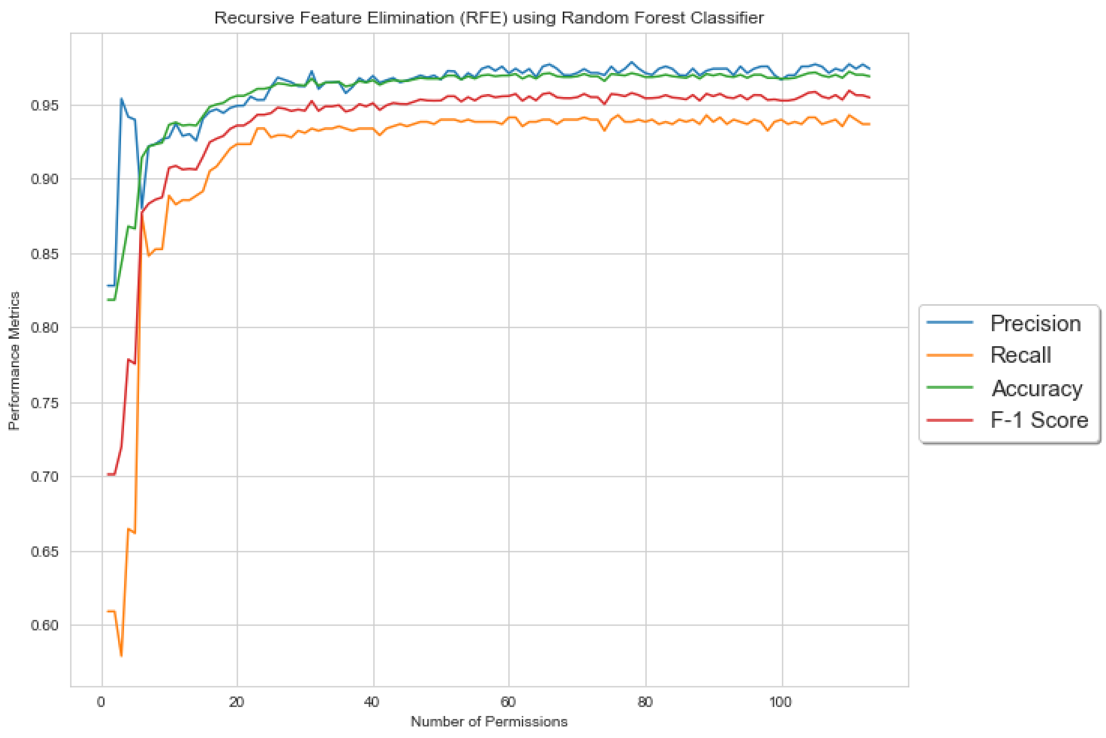


(a) Precision, Recall, Accuracy, F-1 score

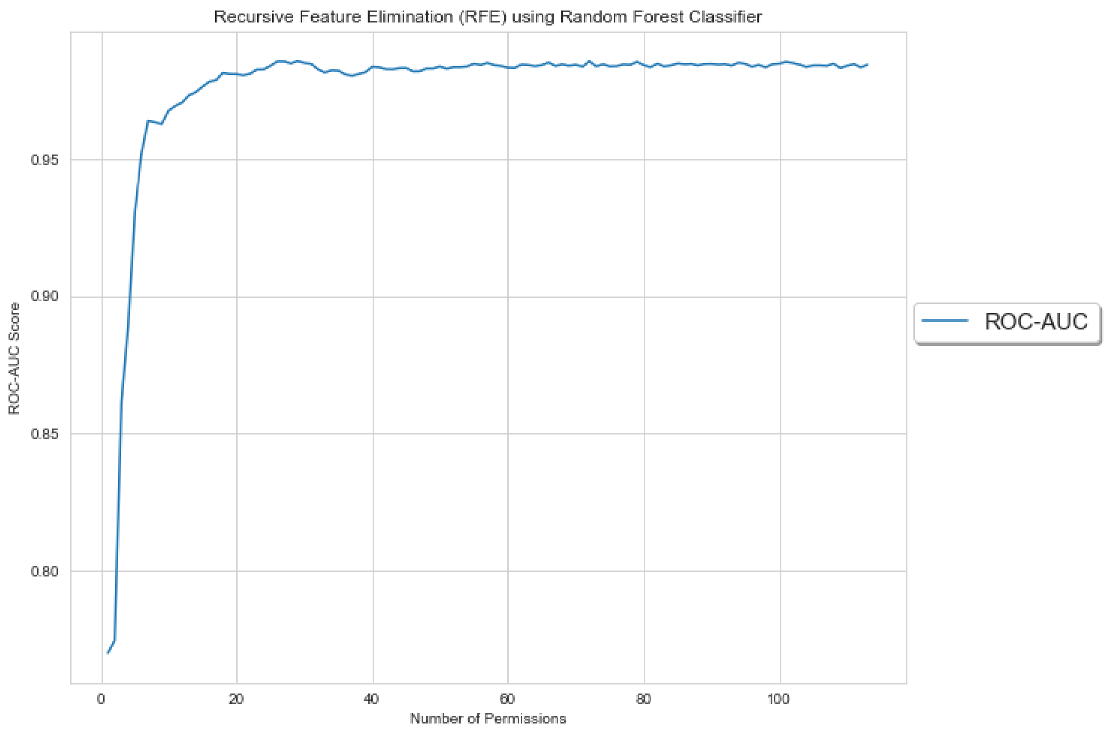


(b) AUC

Figure 5.3: Minimal Range for RFE with Random Forest Classifier for Drebin



(a) Precision, Recall, Accuracy, F-1 score



(b) AUC

Figure 5.4: Minimal Range for RFE with Random Forest Classifier for Malgenome

Table 5.9: Minimal Range of Features (Permissions) using Different Feature Selection Techniques

Feature Selection Technique	**Minimal Range for Drebin**	**Minimal Range for Malgenome**
Mutual Information Gain	63-70	85-90
RFE with Gradient Boosting Classifier	38-42	25-30
RFE with Random Forest Classifier	**34-38**	**23-27**
SelectKBest with chi2	42-47	43-46
SelectFromModel with Random Forest Classifier	35-43	27-32
SelectFromModel with Extra Trees Classifier	40-47	30-35

reduction is Recursive Feature Elimination (RFE) with Random Forest Classifier as it provides the lowest minimal range among other techniques for both data sets. RFE with Gradient Boosting Classifier and SelectFromModel with Random Forest Classifieralso gives a comparatively lower minimal range for both data sets. Other feature selection techniques do not perform equally for both data sets. For instance, Mutual Information Gain and Univariate ROC-AUC scores perform inadequately for both datasets. Univariate ROC-AUC score is not reported here as it fails to reach performance close to all Permissions.

5.4.2 Correlation-based Feature Elimination (CFE) for Permissions

Before proceeding to the performance evaluation, the correlation-based feature elimination (CFE) is performed on the selected minimal range of Permissions. Experimental results show that CFE can reduce the Permissions without limiting the performances. The reduction of Permissions depicted in Table 5.10. For instance, CFE can reduce 34 features to 33 and 23 features to 22 features respectively for the Drebin and Malgenome data sets.

Table 5.10: Correlation-based Feature Elimination (CFE) on Minimal Feature Sets for Permissions

Drebin		**Malgenome**	
Number of Permissions in Minimal Feature Sets	**Reduced Number of Permissions with CFE**	**Number of Permissions in Minimal Feature Sets**	**Reduced Number of Permissions with CFE**
34	33	23	22
35	34	24	23
36	35	25	24
37	36	26	25
38	37	27	26

5.4.3 Performance of Significant Permissions in Malware Detection

Finally, the reduced set of Significant Permissions are evaluated based on the performance metrics, execution time, and comparison with existing works.

Performance Metrics of the Significant Permissions

The performance evaluation for the significant API Calls is based on five metrics. The evaluation is described in Table 5.11 and 5.12 for the minimal range of Permissions. Table 5.11 shows that for the Drebin dataset, the performance metrics using significant Permissions are close to the performance metrics of using all the Permissions (114). Table 5.12 also shows that for the Malgenome dataset, the significant Permissions performs almost identically to the the full feature set of Permissions (113). Specifically, how many significant Permissions should be selected? - It depends on the requirement of the stakeholders. However, it is suggested to use the range of 33-37 significant Permissions based on the performances. The higher the minimal range from the two data sets is recommended to avoid biases.

Table 5.11: Performance Evaluation of Significant Permissions for Drebin

Feature Selection Techniques	Number of Permissions	Reduced Permissions with CFE	Accuracy (%)	Precision (%)	Recall (%)	F1	AUC
None (All Features)	114		96.00	96.51	92.28	0.9433	0.9877
RFE with Random Forest Classifier	34	33	95.46	96.20	91.07	0.9355	0.9838
	35	34	95.38	95.87	91.18	0.9345	0.9843
	36	35	95.66	96.15	91.69	0.9386	0.9840
	37	36	95.70	96.19	91.76	0.9392	0.9843
	38	37	95.82	96.31	91.99	0.9409	0.9840

Table 5.12: Performance Evaluation of Significant Permissions for Malgenome

Feature Selection Techniques	Number of Permissions	Reduced Permissions with CFE	Accuracy (%)	Precision (%)	Recall (%)	F1	AUC
None (All Features)	113	-	96.89	97.41	93.67	0.9547	0.9842
RFE with Random Forest Classifier	23	22	96.58	95.60	94.05	0.9480	0.9863
	24	23	96.74	96.05	94.05	0.9503	0.9864
	25	24	96.68	96.34	93.57	0.9493	0.9888
	26	25	96.92	96.62	94.05	0.9530	0.9891
	27	26	96.97	96.63	94.21	0.9538	0.9893

Execution Time of the Significant Permissions

Table 5.13 shows the comparative execution time of malware detection. The result shows that using the significant Permissions (ranges between 33-37 for Drebin and 22-26 for Malgenome), the execution time of the malware detection is considerably lower than using all the features. For large data sets, this time would be substantially higher.

Table 5.13: Execution Time of Significant Permissions

Drebin		Malgenome	
Number of Permissions	Execution Time (s)	Number of Permissions	Execution Time (s)
114 (All)	11.5	113 (All)	10.9
37	6.67	26	5.76
36	6.45	25	5.47
35	6.43	24	5.39
34	6.13	23	5.32
33	6.02	22	5.24

Comparison with Related Works (Significant Permissions)

Significant Permissions are also evaluated in comparison with several related works that deal with significant/important Permissions. Table 5.14 shows the comparative analysis of the related research. This studyś significant Permissions analysis is regarded as SigPer (22) in the table. Using only 22 significant Permissions for the Malgenome dataset, SigPer (22) performs better than other related works except that SigPIDś precision and F1 score is higher [6]. Although the accuracy and recall of SigPID are less than that of SigPer's (22).

Table 5.14: Comparison with Related Works (Significant Permissions)

Research	**Number of Permissions**	**Accuracy**	**Precision**	**Recall**	**F1**
SigPer (22)	22	**96.57%**	95.60%	**94.05%**	0.9480
SigPID [6]	22	95.63%	**97.54%**	93.62%	**0.9554**
Altaher et al.	24	91%	N/A	N/A	N/A
Wang et al.	40	92.79%	N/A	N/A	N/A

5.4.4 Top Permissions in Android Malware Detection

Finally, the top 40 Permissions are analyzed and reported. According to the Drebin dataset, the lowest minimal range is in between 33-37 Permissions. Thereby, the top 40 Permissions are reported. Table 5.15 shows the top 40 significant Permissions in Malware Detection for the Drebin dataset. These Permissions are derived from the feature selection technique – RFE with Random Forest Classifier. Also, these 40 Permissions are almost identical to the Malgenome dataset except 2 Permissions. More data instances would be conducive to generating a generalizable list of significant Permissions. Yet, as this study primarily suggests an approach for significant Permissions, the dataset to dataset it may slightly vary due to the inconsistency and time of data sets.

Table 5.15: Top Permissions in Android Malware Detection

TOP 40 SIGNIFICANT PERMISSIONS	
SEND_SMS	BLUETOOTH
READ_PHONE_STATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	VIBRATE
RECEIVE_SMS	ACCESS_NETWORK_STATE
READ_SMS	GET_TASKS
USE_CREDENTIALS	SET_WALLPAPER
MANAGE_ACCOUNTS	ACCESS_COARSE_LOCATION
WRITE_SMS	WRITE_SETTINGS
READ_SYNC_SETTINGS	KILL_BACKGROUND_PROCESSES
WRITE_HISTORY_BOOKMARKS	CHANGE_NETWORK_STATE
INSTALL_PACKAGES	CALL_PHONE
CAMERA	READ_LOGS
READ_HISTORY_BOOKMARKS	SYSTEM_ALERT_WINDOW
INTERNET	CHANGE_WIFI_STATE
RECORD_AUDIO	READ_CONTACTS
ACCESS_LOCATION_EXTRA_COMMANDS	ACCESS_WIFI_STATE
MODIFY_AUDIO_SETTINGS	WRITE_EXTERNAL_STORAGE
BROADCAST_STICKY	ACCESS_FINE_LOCATION
WAKE_LOCK	WRITE_SYNC_SETTINGS
RECEIVE_BOOT_COMPLETED	WRITE_APN_SETTINGS

5.5 Significant Ensemble Features Analysis

Aswini et al. [1] suggests that ensemble features (Permissions and API Calls) are more effective in Android malware detection. In assessing the significant ensemble features in Android malware detection, the proposed workflow is carried out. Both Permissions and API Calls are considered in ensemble features analysis. In total 187 (114 Permissions and 73 API Calls) and 184 (113 Permissions and 69 API Calls), ensemble features are analyzed respectively for Drebin and Malgenome data sets. The outcomes and insights from each step of the approach are described in this subsection in order.

5.5.1 Identification of Feature Selection Technique and Minimal Range of Features for Ensemble Features

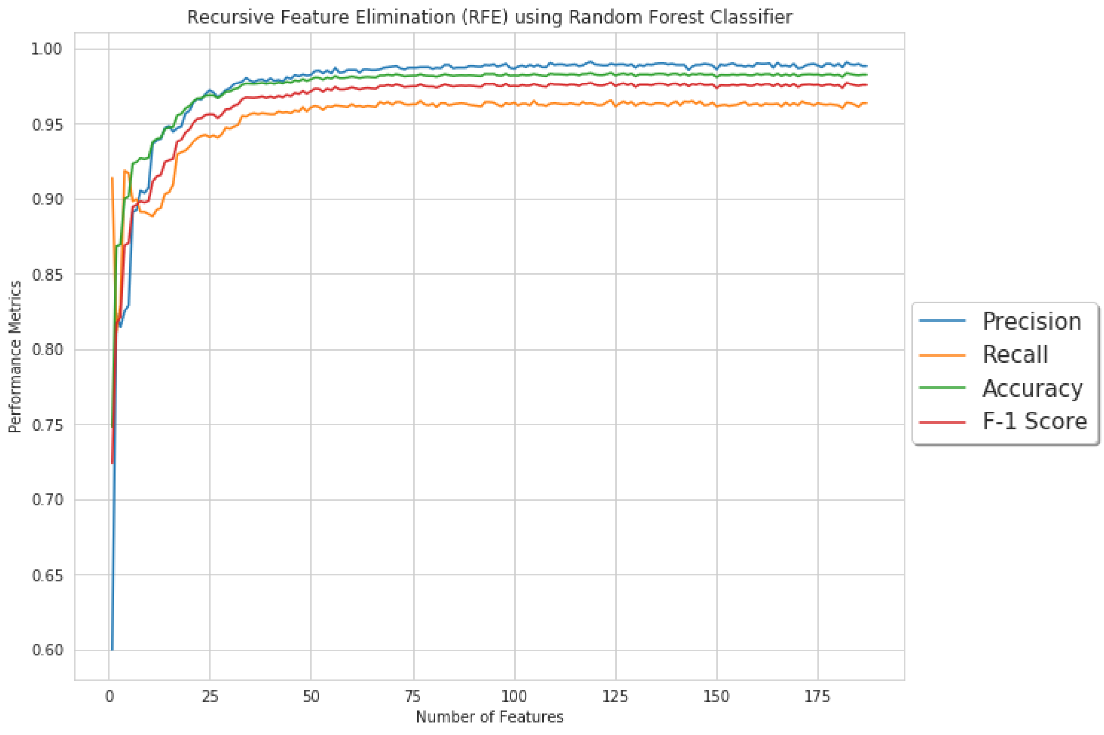
In order to identify the appropriate feature selection technique for malware detection and corresponding minimal range of ensemble features, incremental feature selection techniques are performed for all of the feature selection techniques mentioned in Section 4.4. Plotting the performance metrics with regards to the increasing number of ensemble features assists in identifying the appropriate feature selection technique and the minimal range of ensemble features.

Recursive Feature Elimination (RFE) with the Random Forest classifier provides the minimal range of features for both the Drebin and Malgenome data sets. This can be deduced from Fig (5.5) and (5.6) which represents the Drebin and the Malgenome data sets respectively.

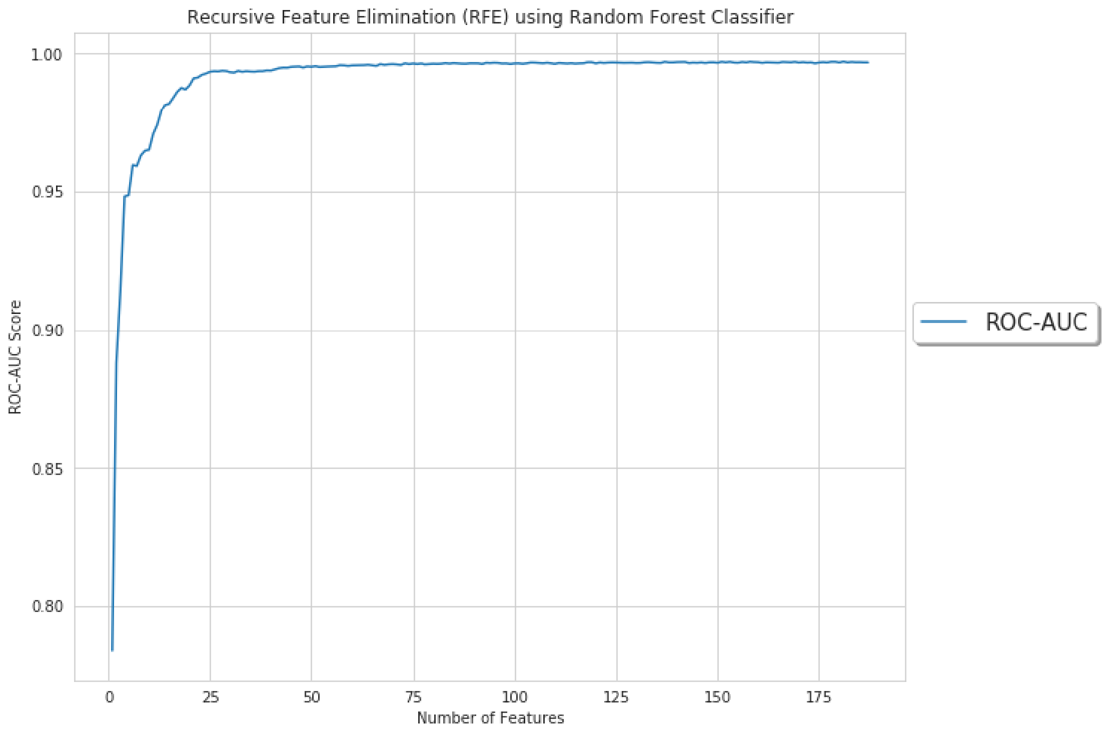
For the Drebin dataset, according to the Fig (5.5), as the number of ensemble features is increased, the performances (Accuracy, Precision, Recall, F1 score, AUC) are also increased initially. However, in the range between 30-35 ensemble features (approximately), the performance metrics are getting stable and remain constant for the rest of the ensemble features. The performance lines remain horizontal with respect to the number of ensemble features from the minimal range of features. So, it can be inferred that by taking those 30-35 ensemble features, the performances are close enough to the actual performances of all the 187 ensemble features.

Similar phenomena can be deduced from Figure (5.6). In the case of the Malgenome dataset, from Figure (5.6), it can be inferred that the minimal range of features is only 15-20 ensemble features approximately. After that range, the performances remain stable practically. Malware detection performances within the range are similar to taking all of the 184 ensemble features.

The minimal ranges for all the feature selection techniques are depicted in Table 5.16. From the experiments, the best feature selection technique for ensemble features reduction is Recursive Feature Elimination (RFE) with Random Forest Classifier as it provides the lowest minimal range among other techniques for both data sets. RFE with Gradient Boosting Classifier and SelectFromModel with Random Forest and Extra Trees Classifier also gives a comparatively lower minimal

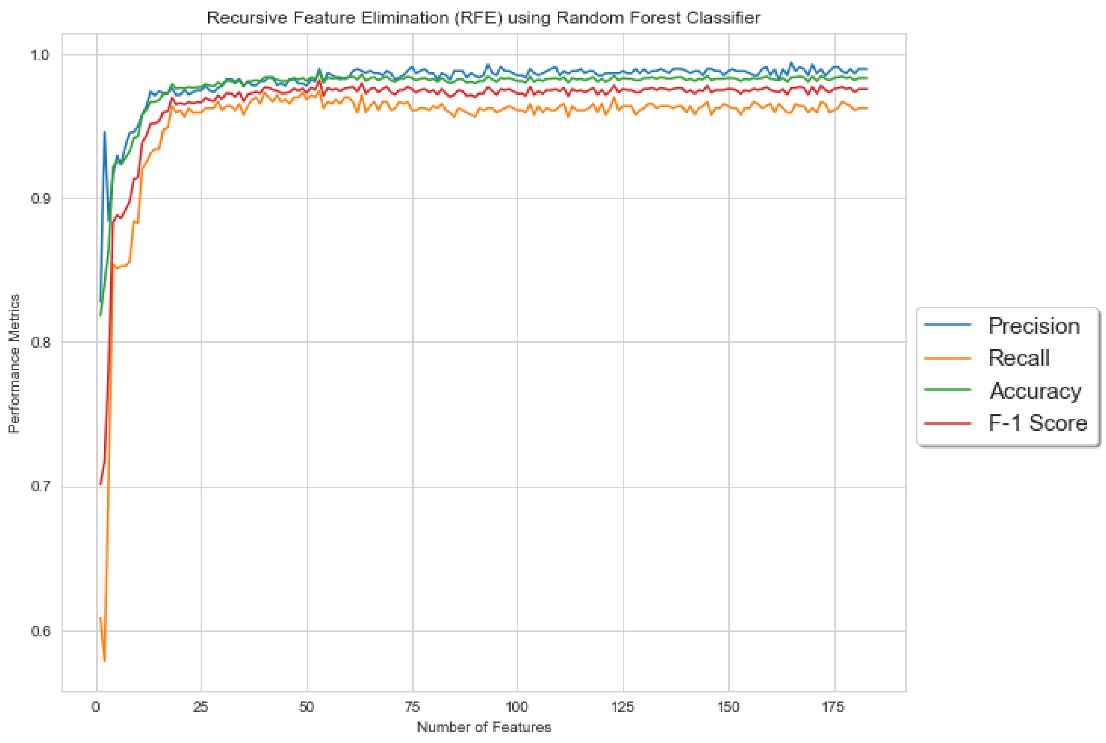


(a) Precision, Recall, Accuracy, F-1 score

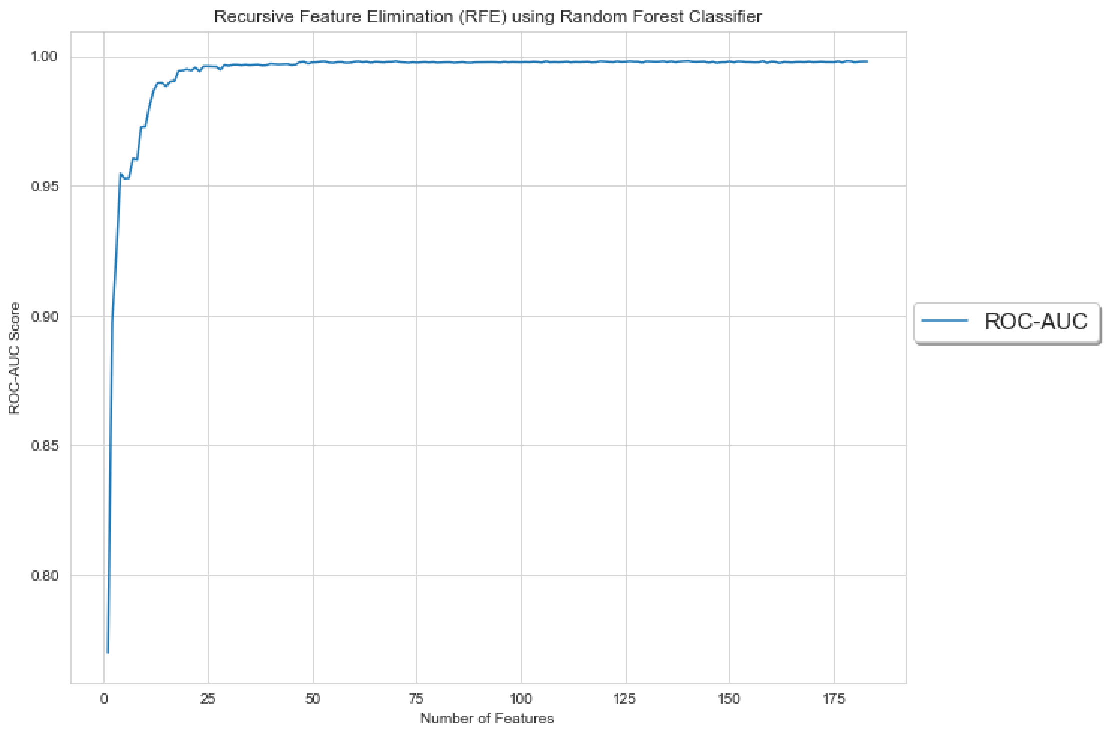


(b) AUC

Figure 5.5: Minimal Range for RFE with Random Forest Classifier for Drebin



(a) Precision, Recall, Accuracy, F-1 score



(b) AUC

Figure 5.6: Minimal Range for RFE with Random Forest Classifier for Malgenome

Table 5.16: Minimal Range of Features (Ensemble Features) using Different Feature Selection Techniques

Feature Selection Technique	**Minimal Range for Drebin**	**Minimal Range for Malgenome**
Mutual Information Gain	60-70	52-57
RFE with Gradient Boosting Classifier	32-37	25-32
RFE with Random Forest Classifier	**30-35**	**15-20**
SelectKBest with chi2	50-55	37-43
SelectFromModel with Random Forest Classifier	37-45	25-35
SelectFromModel with Extra Trees Classifier	38-45	17-30

range for both data sets. Other feature selection techniques do not perform equally well for both data sets. For instance, Mutual Information Gain performs poorly for both data sets.

Table 5.17: Correlation-based Feature Elimination (CFE) on Minimal Feature Sets for Ensemble Features

Drebin		**Malgenome**	
Number of API Calls in Minimal Feature Sets	**Reduced Number of API Calls with CFE**	**Number of API Calls in Minimal Feature Sets**	**Reduced Number of API Calls with CFE**
20	17	17	15
21	18	18	16
22	19	19	16
23	20	20	19
24	21	21	20
25	21	22	21

5.5.2 Correlation-based Feature Elimination (CFE) for Ensemble Features

Before proceeding to the performance evaluation, the correlation-based feature elimination (CFE) is performed on the selected minimal range of ensemble features. Experimental results show that CFE can reduce ensemble features without limiting the performances. The reduction of ensemble features is depicted in Table

5.17. For instance, CFE can reduce 35 features to 30 and 15 features to 12 features respectively for the Drebin and Malgenome data sets.

5.5.3 Performance of Significant Ensemble Features in Malware Detection

Finally, the reduced set of Significant ensemble features are evaluated based on the performance metrics, execution time, and comparison with existing works.

Performance Metrics of the Significant Ensemble Features

The performance evaluation for the significant API Calls is based on five metrics. The evaluation is described in Table 5.18 and 5.19 for the minimal range of ensemble features. Table 5.18 shows that for the Drebin dataset, the performance metrics using significant ensemble features are close to the performance metrics of using all the ensemble features (187). Table 5.19 also shows that for the Malgenome dataset, the significant ensemble features perform almost identically to the the full feature set of ensemble features (184). Specifically, how many significant ensemble features should be selected? - It depends on the requirement of the stakeholders. However, it is suggested to use the range of 30-35 significant ensemble features based on the performances. The higher the minimal range from the two data sets is recommended to avoid biases.

Table 5.18: Performance Evaluation of Significant Ensemble Features for Drebin

Feature Selection Techniques	**Number of Ensemble Features**	**Reduced Features with CFE**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1**	**AUC**
None (All Features)	187	-	98.27	98.84	96.36	0.9758	0.9968
RFE with Random Forest Classifier	30	25	97.19	97.23	94.96	0.9607	0.9933
	31	26	97.29	97.27	95.18	0.9620	0.9935
	32	27	97.31	97.34	95.18	0.9624	0.9935
	33	28	97.53	97.53	95.59	0.9654	0.9935
	34	29	97.59	97.79	95.51	0.9663	0.9933
	35	30	97.53	97.61	95.51	0.9654	0.9934

Table 5.19: Performance Evaluation of Significant Ensemble Features for Malgenome

Feature Selection Techniques	Number of Ensemble Features	Reduced Features with CFE	Accuracy (%)	Precision (%)	Recall (%)	F1	AUC
None (All Features)	184	-	98.32	98.95	96.24	0.9755	0.9980
RFE with Random Forest Classifier	15	12	96.68	97.68	92.78	0.9514	0.9885
	16	12	97.42	98.20	94.43	0.9624	0.9900
	17	12	97.26	97.59	94.58	0.9603	0.9884
	18	13	97.74	97.65	95.94	0.9673	0.9939
	19	15	97.58	97.01	96.09	0.9652	0.9951
	20	15	97.32	96.58	95.78	0.9614	0.9944

Execution Time of the Significant Ensemble Features

Table 5.20 shows the comparative execution time of malware detection. The result shows that using the significant ensemble features (ranges between 25-30 for Drebin and 12-15 for Malgenome), the execution time of the malware detection is notably lower than using all the features. For large data sets, this time would be substantially higher.

Table 5.20: Execution Time of Significant Ensemble Features

Drebin		Malgenome	
Number of Ensemble Features in Minimal Feature Sets	Execution Time (s)	Number of Ensemble Features in Minimal Feature Sets	Execution Time (s)
187	17.8	184	16.6
30	5.45	15	3.34
29	5.32	13	3.21
28	4.96	12	3.14
27	5.01		
26	4.78		
25	4.56		

Comparison with Related Works (Significant Ensemble Features)

Significant ensemble features are also evaluated in comparison with two related works that deal with significant/important ensemble features. Table 5.21 shows the comparative analysis of the related research. This studyś significant ensemble features analysis are regarded as SigEnsemble (13) and SigEnsemble (29) in the table. SigEnsemble (13) refers to 13 significant ensemble features from the Malgenome dataset, and SigEnsemble (29) refers to 29 significant ensemble features from the Drebin dataset. SigEnsemble (13) and SigEnsemble (29) both outperform the other two research except Aswini et al. [?] achieves higher recall. Although Aswini et al. use 168 ensemble features while this study uses 13 and 29 ensemble features respectively for Malgenome and Drebin data sets.

Table 5.21: Comparison with Related Works (Significant Ensemble Features)

Research	**Number of Ensemble Features**	**Accuracy**	**Precision**	**Recall**	**F1**
SigEnsemble (13)	13	**97.74**	**97.65**	95.94	**0.9673**
SigEnsemble (29)	29	**97.59**	**97.79**	95.51	**0.9663**
Aswini et al.	168	93.02	N/A	N/A	N/A
Zhao et al.	398	97.5	N/A	**97.5**	N/A

5.5.4 Top Ensemble Features in Android Malware Detection

Finally, the top 30 ensemble features are analyzed and reported. According to the Drebin dataset, the lowest minimal range is in between 25-30 ensemble features. Thereby, the top 30 ensemble features are reported. Table 5.22 shows the top 30 significant ensemble features in Malware Detection for the Drebin dataset. These ensemble features are derived from the feature selection technique – RFE with Random Forest Classifier. Also, these 30 ensemble features are almost identical to the Malgenome dataset except 4 ensemble features. More data instances would be conducive to generating a generalizable list of significant ensemble features. Yet, as this study primarily suggests an approach for significant ensemble features, the dataset to dataset it may slightly vary due to the inconsistency and time of data sets.

Table 5.22: Top Ensemble Features in Android Malware Detection

TOP 30 SIGNIFICANT ENSEMBLE FEATURES	
transact	RECEIVE_SMS
onServiceConnected	Ljava.lang.Class.getDeclaredField
bindService	READ_SMS
attachInterface	getCallingUid
ServiceConnection	Ljavax.crypto.spec.SecretKeySpec
SEND_SMS	USE_CREDENTIALS
Ljava.lang.Class.getCanonicalName	MANAGE_ACCOUNTS
Ljava.lang.Class.getMethods	TelephonyManager.getLine1Number
Ljava.lang.Class.cast	DexClassLoader
Ljava.net.URLDecoder	HttpGet.init
android.telephony.SmsManager	SecretKey
READ_PHONE_STATE	Ljavax.crypto.Cipher
Landroid.content.Context.registerReceiver	WRITE_SMS
Ljava.lang.Class.getField	READ_SYNC_SETTINGS
GET_ACCOUNTS	AUTHENTICATE_ACCOUNTS

5.6 Summary

This chapter provides the evaluation of significant features in Android malware detection. It evaluates significant API Calls, significant Permissions, and significant ensemble features separately. Experimental results suggest that analyzing significant features not only reduces complexity but also maintains performance as close to all features. This study is assessed on two benchmark data sets to secure generalizability. For the Drebin dataset, it reduces 73 API Calls to 17-21 API Calls, 114 Permissions to 33-37 Permissions, and 187 ensemble features to 25-30 features while retaining performance. For the Malgenome dataset, it reduces 69 API Calls to 15-21 API Calls, 113 Permissions to 22-26 Permissions, and 184 ensemble features to 12-15 features while retaining performance. Altogether, this study manages to identify significant features for Android malware detection while maintaining detection performance. It enables large scale application of this approach with a feasible strategy.

perhaps an appropriate statistical significance test was in order.

Chapter 6

Conclusion

In this study, an approach is proposed for analyzing and identifying significant features in Android malware detection. In short, the proposed approach exhibits promises, performs proficiently, and outperforms most of the related works. This chapter sums up the study and ends by presenting future works.

6.1 Discussion

This study deals with examining features for reducing the irrelevant ones without tampering significant features. It presents an approach for significant features identification. Significant features are analyzed from three aspects, such as significant API Calls analysis, significant Permissions analysis, and significant ensemble features analysis. Primarily, it incrementally employs several feature selection techniques to determine the best-suited feature selection algorithm and the minimal number of significant features. Analytical results show that RFE with Random Forest Classifier provides the minimal range of significant features for all cases. Further, by incorporating a correlation-based feature elimination strategy, it reduces the minimal range of significant feature sets more.

Experimental results show that significant features perform adeptly with regards to all features set. Also, in terms of run time performance and in comparison with related research, significant features turn out to be more effective. Lastly, significant ensemble features outperform individual types of features, like Permissions and API Calls. The figures and tables of Chapter 5 show that phenomenon more clearly.

If the experiments were not repeated the change may have happened by chance. The experiments have, however, been replicated with two different data sets to ensure that the progress demonstrated by the adaptation technique is not simply by chance.

6.2 Threats to Validity

This section provides the threats that can hamper the accuracy and effectiveness of the proposed model. Threats to internal and external validity have been identified which are given below.

6.2.1 Threats to Internal Validity

The parameters of different techniques and algorithms are susceptible to bias and can be examined differently by different analysts. For instance, the parameters of feature selection algorithms, like tree size of Random Forest, the threshold of ROC-AUC scores, correlation threshold, etc. arise threats to internal validity. The metric threshold has to be picked carefully. The proposed approach will not provide a similar performance if the threshold values are changed. It is worth mentioning that there is not any general consensus on the threshold values, it varies case to case.

Besides, the execution time measurement is susceptible to bias and can be turned to be different for other machines. Although the execution time measurement is carried out five times and the mean value is taken.

6.2.2 Threats to External Validity

In this study, only the Drebin and Malgenome data sets have been analyzed, which is subject to bias and lack of generalizability, threatening external validity. All the imprecision and errors in the dataset would impact the results of this study.

The evaluation and findings depend on some third-party systems such as AndroPyTool, Weka, Jupyter Notebook, etc. If either of these includes inaccuracies, the experimental results would also be impacted.

6.3 Challenges and Opportunities in Android Malware Detection

Almost 10 million new malware are found each month. But there does not exist any up-to-date dataset of malware. So, their performance in malware detection is doubtful considering the vast population of the new malware. Dataset inadequacy is a vital factor as an appropriate dataset is required for research evaluation. Therefore, it provokes escalating challenges as well as opportunities for new researchers. Malware data sets should be updated on a regular basis to assure the effectiveness of the new research and to justify the feasibility of the existing research. Modern data extraction tools for Android, such as Androdata, ApkTool, Droidbox, Androguard, etc. can efficiently extract static and dynamic data. Using those tools and collaborating with benign and malware apps sources (e.g., Google Play Store, VirusShare, etc.), data inadequacy can be reduced.

As existing malwares behavior is decoded by the existing tool or research outcome; malware authors update existing malware families and create new malware families frequently to evade detection. They try to trick existing detection systems by introducing new behavior as well as exhibiting benign behavior. Numerous features and malware families provoke much complexity which may limit and challenge the progression of Android malware detection. Consequently, malware detection becomes more challenging.

Most of the existing research only deals with some common features. But it is more likely that there exist more distinguishable features. For instance, Kabakus et al. [67] reveal many unknown characteristics of Android malware, however, it did not integrate any machine learning technique to detect malware. They reveal that over-privileged permissions are one of the characteristics of malware. Besides, they uncover that malware's average number of incoming and outgoing connections, the average size of download and upload, etc. are distinguishable features in malware. It suggests that there will be decent opportunities in exploring new features.

A lot of opportunities and research directions are available right now. Researchers' enthusiastic focus on this field would have been beneficial to fight against the rising malware authors community.

6.4 Future Work

In future work, the approach will be evaluated using different and large data sets. That will ensure its applicability on large scale and omit biases arisen from data sets.

Apart from the Permissions and API Calls, other static and dynamics features will be incorporated for better performance. Features like app metadata, intents, network activities, system call, file operations, etc. will be used in future work.

Besides, looking for more discernible features would be a promising research direction. Many leading-edge feature selection techniques can be used to explore new features and reduce the complexity as there are numerous features incidentally. Reinforcement learning, Deep learning, Bagging, Boosting, Tree-based, and embedded feature selection techniques can be used for reducing complexity and discernible features exploration.

Moreover, finding new malware families can be a promising research direction as malware family is growing exceedingly. How do we detect new malware families effectively? - would be prospective future research in this regard.

Bibliography

[1] A. Aswini and P. Vinod, "Android malware analysis using ensemble features," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 303–318, Springer, 2014.

[2] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for android malware detection," in *2015 IEEE symposium on computers and communication (ISCC)*, pp. 714–720, IEEE, 2015.

[3] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *International conference on security and privacy in communication systems*, pp. 86–103, Springer, 2013.

[4] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, "Quick and accurate android malware detection based on sensitive apis," in *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pp. 143–148, IEEE, 2018.

[5] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and api calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.

[6] L. Sun, Z. Li, Q. Yan, W. Srisa-an, and Y. Pan, "Sigpid: significant permission identification for android malware detection," in *2016 11th international conference on malicious and unwanted software (MALWARE)*, pp. 1–8, IEEE, 2016.

[7] A. Altaher and O. BaRukab, "Android malware classification based on anfis with fuzzy c-means clustering using significant application permissions," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 25, no. 3, pp. 2232–2242, 2017.

[8] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.

[9] Techeconomy, "Android: Market share and other interesting stats [infographic]," Jul 2019.

[10] B. Popper, "Google announces over 2 billion monthly active devices on android," May 2017.

[11] C. Lueg, "Cyber attacks on android devices on the rise," Nov 2018.

[12] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–41, 2017.

[13] M. Atkinson, "App permissions: An analysis of android phones," Aug 2020.

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

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

[16] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*, pp. 121–128, IEEE, 2013.

[17] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pp. 37–42, IEEE, 2014.

[18] J. Fernando, "What is an api ? how to call an api from android ?." `https://droidmentor.com/api-call-api-android/`, Oct 2016. Online; accessed 14 December 2019.

[19] B. Popper, "Google announces over 2 billion monthly active devices on android." `https://www.theverge.com/2017/5/17/15654454/android-reaches-2-billion-monthly-active-users`. Online; accessed 12 October 2019.

[20] C. Lueg, "Cyber attacks on android devices on the rise." `https://rb.gy/blxoit`. Online; accessed 13 October 2019.

[21] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Generation Computer Systems*, vol. 97, pp. 887–909, 2019.

[22] "Permissions overview : Android developers." `https://developer.android.com/guide/topics/permissions/overview`. Online; accessed 26 October 2019.

[23] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *2012 Seventh Asia Joint Conference on Information Security*, pp. 62–69, IEEE, 2012.

[24] "App permissions best practices : Android developers." `https://developer.android.com/training/permissions/usage-notes`. Online; accessed 13 December 2019.

[25] "Users sdk element: Android developers." `https://developer.android.com/guide/topics/manifest/uses-sdk-element#ApiLevels`. Online; accessed 6 June 2020.

[26] "Android api levels." `http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/appendix/api-levels.html`. Online; accessed 6 June 2020.

[27] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[28] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *2013 IEEE 25th international conference on tools with artificial intelligence*, pp. 300–305, IEEE, 2013.

[29] X. Su, D. Zhang, W. Li, and K. Zhao, "A deep learning approach to android malware feature learning and detection," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 244–251, IEEE, 2016.

[30] R. Raphael, P. Vinod, and B. Omman, "X-anova ranked features for android malware analysis," in *2014 Annual IEEE India Conference (INDICON)*, pp. 1–6, IEEE, 2014.

[31] S. Feldman, D. Stadther, and B. Wang, "Manilyzer: automated android malware detection through manifest analysis," in *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pp. 767–772, IEEE, 2014.

[32] K. Dhanya and K. T. Gireesh, "Efficient android malware scanner using hybrid analysis," *International Journal of Recent Technology and Engineering (TM)*, vol. 7, pp. 76–80, 2019.

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

[34] W.-C. Wu and S.-H. Hung, "Droiddolphin: a dynamic android malware detection framework using big data and machine learning," in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, pp. 247–252, 2014.

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

[36] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[37] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *2015 IEEE 39th annual computer software and applications conference*, vol. 2, pp. 422–433, IEEE, 2015.

[38] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "Samadroid: a novel 3-level hybrid malware detection model for android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[39] F. Martinelli, F. Mercaldo, A. Saracino, and C. A. Visaggio, "I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pp. 129–136, IEEE, 2016.

[40] A. Martín, 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.

[41] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[42] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "Hadm: Hybrid analysis for detection of malware," in *Proceedings of SAI Intelligent Systems Conference*, pp. 702–724, Springer, 2016.

[43] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[44] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1808–1815, 2013.

[45] A. Kapratwar, F. Di Troia, and M. Stamp, "Static and dynamic analysis of android malware.," in *ICISSP*, pp. 653–662, 2017.

[46] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile android applications," in *2016 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 155–156, IEEE, 2016.

[47] K. Patel and B. Buddadev, "Detection and mitigation of android malware through hybrid approach," in *International Symposium on Security in Computing and Communication*, pp. 455–463, Springer, 2015.

[48] M. Yusof, M. M. Saudi, and F. Ridzuan, "Mobile botnet classification by using hybrid analysis," *International Journal of Engineering & Technology*, vol. 7, no. 4.15, pp. 103–108, 2018.

[49] A. Martín, R. Lara-Cabrera, and D. Camacho, "A new tool for static and dynamic android malware analysis," *Data Science and Knowledge Engineering for Sensing Decision Support*, pp. 509–516, 2018.

[50] A. Kraskov, H. Stögbauer, and P. Grassberger, "Erratum: estimating mutual information [phys. rev. e 69, 066138 (2004)]," *Physical Review E*, vol. 83, no. 1, p. 019903, 2011.

[51] B. C. Ross, "Mutual information between discrete and continuous data sets," *PloS one*, vol. 9, no. 2, p. e87357, 2014.

[52] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[53] "sklearn.feature˙selection.rfe." `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html`. Online; accessed 9 January 2020.

[54] M. B. Fraj, "In depth: Parameter tuning for gradient boosting," Dec 2017.

[55] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[56] "sklearn.feature˙selection.selectkbest." `https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html`. Online; accessed 12 January 2020.

[57] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[58] "1.13. feature selection." `https://scikit-learn.org/stable/modules/feature_selection.html`. Online; accessed 12 January 2020.

[59] "11. feature engineering for machine learning — data science beginners." https://datasciencebeginners.com/2018/11/26/11-feature-engineering-for-machine-learning/. Online; accessed 13 January 2020.

[60] "Basic concepts of correlation." http://www.real-statistics.com/correlation/basic-concepts-correlation/. Online; accessed 16 January 2020.

[61] E. Solutions and . Name, "Accuracy, precision, recall f1 score: Interpretation of performance measures," Nov 2016.

[62] S. Narkhede, "Understanding auc - roc curve," May 2019.

[63] A. Martín, R. Lara-Cabrera, and D. Camacho, "A new tool for static and dynamic android malware analysis," *Data Science and Knowledge Engineering for Sensing Decision Support*, pp. 509–516, 2018.

[64] "Weka." http://www.cs.waikato.ac.nz/ml/weka/.

[65] "Project jupyter."

[66] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE symposium on security and privacy*, pp. 95–109, IEEE, 2012.

[67] A. T. Kabakus and I. A. Dogru, "An in-depth analysis of android malware using hybrid techniques," *Digital Investigation*, vol. 24, pp. 25–33, 2018.