Hindawi
Security and Communication Networks
Volume 2021, Article ID 7035233, 22 pages
https://doi.org/10.1155/2021/7035233

Research Article

Dominant Feature Selection and Machine Learning-Based Hybrid Approach to Analyze Android Ransomware

Tanya Gera,[1] **Jaiteg Singh**,[1] **Abolfazl Mehbodniya**,[2] **Julian L. Webber**,[3] **Mohammad Shabaz**,[4,5] **and Deepak Thakur**[1]

[1]*Chitkara University Institute of Engineering and Technology, Chitkara University, Chandigarh, Punjab, India*
[2]*Department of Electronics and Communication Engineering, Kuwait College of Science and Technology, Kuwait, Kuwait*
[3]*Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, Japan*
[4]*Arba Minch University, Arba Minch, Ethiopia*
[5]*Department of Computer Science Engineering, Chandigarh University, Punjab, India*

Correspondence should be addressed to Tanya Gera; tanya.gera@chitkara.edu.in, Jaiteg Singh; jaiteg.singh@chitkara.edu.in, and Mohammad Shabaz; mohammad.shabaz@amu.edu.et

Received 26 August 2021; Revised 20 September 2021; Accepted 1 October 2021; Published 9 November 2021

Academic Editor: Jie Cui

Ransomware is a special malware designed to extort money in return for unlocking the device and personal data files. Smartphone users store their personal as well as official data on these devices. Ransomware attackers found it bewitching for their financial benefits. The financial losses due to ransomware attacks are increasing rapidly. Recent studies witness that out of 87% reported cyber-attacks, 41% are due to ransomware attacks. The inability of application-signature-based solutions to detect unknown malware has inspired many researchers to build automated classification models using machine learning algorithms. Advanced malware is capable of delaying malicious actions on sensing the emulated environment and hence posing a challenge to dynamic monitoring of applications also. Existing hybrid approaches utilize a variety of features combination for detection and analysis. The rapidly changing nature and distribution strategies are possible reasons behind the deteriorated performance of primitive ransomware detection techniques. The limitations of existing studies include ambiguity in selecting the features set. Increasing the feature set may lead to freedom of adept attackers against learning algorithms. In this work, we intend to propose a hybrid approach to identify and mitigate Android ransomware. This study employs a novel dominant feature selection algorithm to extract the dominant feature set. The experimental results show that our proposed model can differentiate between clean and ransomware with improved precision. Our proposed hybrid solution confirms an accuracy of 99.85% with zero false positives while considering 60 prominent features. Further, it also justifies the feature selection algorithm used. The comparison of the proposed method with the existing frameworks indicates its better performance.

1. Introduction

Ransomware has blown away the cyber security world in recent past. It targets the major losses like data, money, and even life. These are special malware used to extort money in return of access and data without user's consent. Attackers are consistently working on producing advanced methods to deceit the victim and generate revenue. According to coalition's cyber insurance claim report (Cyber Insurance Claims Report, 2020), out of 87% reported attacks, 41% are due to ransomware attacks as shown in Figure 1. The possible reason for this significant increase is because of COVID-19 pandemic; most of the employees are working remotely. The rapidly changing nature and distribution strategies along with smart tactics are also responsible for deteriorated performance of primitive ransomware detection techniques. Ransomware is generally seen in two forms: locker-ransomware and crypto-ransomware [1]. Locker-ransomware attacks lock the victim's device to restrict its use until they pay ransom. On the other side, crypto-ransomware attacks encrypt all personal files to make them inaccessible for owner. Victims are forced to pay ransom to allow unrestricted access to their own personal and confidential data. To classify, analyze, and detect malicious application

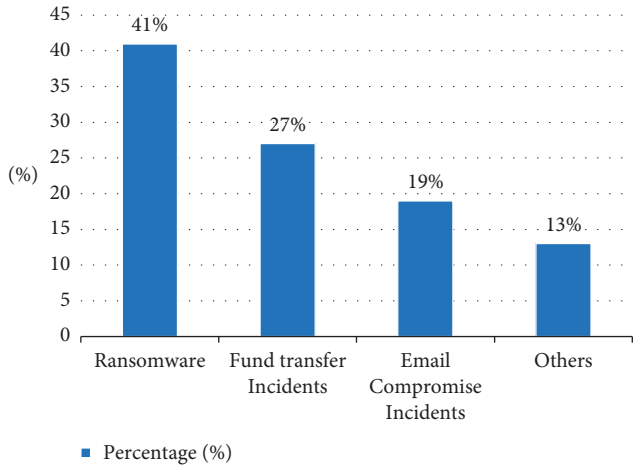


Figure 1: Ransomware share in recent reported cyber incidents (modified from Cyber Insurance Claims Report, 2020).

samples, there exists use of two primitive approaches, i.e., static and dynamic techniques. Static techniques examine the applications by matching their signature, code, or permissions used and can detect previously known ransomware only.

Though the literature witnessed that static analysis is fast and effective in detection of Android ransomware, static analysis techniques are popular for their ability of identifying only known ransomware. Considering fast-evolving nature of Android ransomware, static analysis is not enough. Static analysis is based on code and signature similarity and fails at code obfuscation. On the other hand, dynamic analysis checks the general behavior of an application while execution. Dynamic analysis techniques are strong enough to withstand with vulnerable situations and can even detect suspicious behavior even when code is compressed or encrypted. However, dynamic analysis also has a few flaws against smart malware tactics being used these days. Smart malware actions are sometimes triggered only under certain conditions, which is not possible to achieve in emulated testing environment. Hence, fusion of effective static techniques with dynamic techniques could give a robust hybrid solution for Android ransomware. The literature also states that there exist comparatively less studies based on hybrid technique for identifying Android ransomware. Existing hybrid solutions majorly vary in feature set used for detection of Android ransomware. Most of the hybrid approaches focus on a specific ransomware family or a specific ransomware type or specific feature only. Those type-specific or family-specific solutions would be difficult to consider as a generalized solution. Another important aspect to be considered here is novelty required in collecting and utilizing the important features for analysis. Ransomware families utilized new evolving features for constructing new variants, hence creating need of constructing robust feature set for analysis and detection. The success of any approach directly depends on feature set and feature selection method being used. To accurately classify the applications, the feature set being used has to be well-built. Extracting prominent features and feature selection methods to be used is an ongoing research challenge. Suspicious authors constantly modify a few features to make frequent new variants, hence posing challenge for existing techniques. However, most of the existing studies focus on one or two types of features only for their analysis and detection while testing its run-time behavior. Though system calls, permissions and APIs are important features to be used for analysis and detection of Android ransomware. However, the literature lacks in kernel level checks, file operations, system component, phone state, and so on. Researcher often faces difficulty in predicting all possible behavior set due to limited availability of ransomware dataset and its fast-evolving nature.

In this work, we performed static analysis as well as dynamic analysis over the collected sample of 3249 clean and malicious applications. Static analysis was performed using Apk tool. In the static feature extraction phase, we focus on manifest file to extract permissions associated with the application sample. In parallel, dynamic analysis was performed over the collected data samples using an emulator, i.e., habo analysis system. During the dynamic feature extraction phase, we focussed on API calls, system calls, permissions, file operations, network features, and other system components. Further, static and dynamic feature vectors were transformed to build combined feature matrix containing unique features. A novel feature selection algorithm was applied to select k-prominent features iteratively. Multiple machine learning classifiers were applied to classify samples as clean or ransomware.

The major contributions of this work are follows:

(i) This work demonstrated the effective use of obtained dynamic features by studying combined impact of all the dynamic features. To the best of our knowledge, prior existing studies utilized one or two standard features like system calls and API. Here, we focussed on all the significant obtained dynamic features to build the efficient dynamic model.

(ii) We have also built a dominant feature selection algorithm to extract top k-dominant features being used by Android ransomware samples and clean sample. This helped to discriminate among risky and nonrisky features to effectively analyze malicious behavior.

(iii) With exhaustive experimentation by varying the number of features to be 20, 40, 60, and till 80, we showed the absolute difference in nominal frequency of features used by Android ransomware and clean applications.

(iv) We evaluated the effectiveness of machine learning classifiers by calculating accuracy, false positive, and false negative rate of each classifier for different set of features iteratively. The result shows that among all the machine learning algorithms, random forest algorithms achieved the highest accuracy of about 99.85% with zero false negative.

(v) We have also compared the results of our proposed method with those of the existing system as shown in Table 1. The results of our proposed hybrid

Table 1: Comparison with existing studies.

Reference	Approach	Machine learning model used	Feature set used	Accuracy (%)
[2]	Static	Random forest, logistic regression, XGBoost, Naive Bayes, support vector machine (SVM), deep learning, and decision tree classifier	Intent, permission, API calls, system commands, and malicious activities	96.3
[3]	Dynamic	Naive Bayes, SVM, and logistic regression	Application programming interface (API)	97
[4]	Hybrid	SVM	Permission, API calls, system calls	99.7
	Our proposed method (dynamic)	J48, LMT, random forest, and random tree	System calls, system components, system command, phone events, run-time permissions, and broadcast receivers	99.85

framework outperform the existing static, dynamic, and hybrid approaches. Our proposed hybrid solution confirms the accuracy of 99.85% with zero false positives while considering 60 prominent features. Further, it also justifies the feature selection algorithm used.

The rest of the paper is organized as follows. The second section presents related work. The complete methodology followed is explained in the third section. The fourth section presents experimental results followed by conclusion and future scope in the last section.

2. Related Work

Two prominent approaches to restrict ransomware infections are static and dynamic analysis of software applications. Static analysis investigates the structural properties of an application without executing it. It primarily emphasizes on code, metadata, and digital signatures imbued within software [5–7]. On the contrary, dynamic analysis examines application behavior by executing it. It executes software within a simulated environment and studies its behavior. Application behavior corresponds to the access permissions, network usage, and information shared, processed, and exchanged by the software application during execution. Static analysis requires less resources and is fast. However, they got failed in case of code obfuscation. On the other hand, dynamic approaches are more effective in performing actual behavior check. However, dynamic approaches are incapable of executing all possible paths and also cannot check interapplication communication on emulators. Hence, many researchers have also worked on hybrid approaches to increase the performance of ransomware detection.

Reference [8] attained lot of popularity and success because in their methodology, they make use of multiple properties together for the analysis and detection. They used source code as well as permissions for capturing static features. This model achieved better performance results by exploring feature level granularity through API calls. Reference [9] proposed that a significant static approach developed was based on application features for detection of malware. It captures important permissions and suspicious API calls of applications, assigns a weight value to them, and then compares it with a threshold value so as to make appropriate decisions. Weight value for each application is based on the nature of the identified malicious patterns. Reference [10] gave a signature-based static technique. Its aim was to scan the payload to check the threatening strings relevant to financial claim. However, this technique was not much popular because generally text messages for financial claim are sent from C&C (Command and Control) server. Reference [2] proposed framework consists of multiple layers for filtrations. In this paper, they generate a message digest value, i.e., MD5 (message digest) based on suspicious permission being used, dangerous permissions being granted, and hazardous intentions. Appropriate decisions are further made on basis of hash value comparison. Reference [3] focussed on checking whether any file had undergone any remarkable changes. Authors make use of techniques like content similarity and entropy measurement for performing the checks. Reference [4] framed a static model capable of identifying both locker as well as crypto-ransomware. This model does not require any apk to be decompiled because its detection is based on bytecode of application. It does not make use of source code. It also can detect the multiple variants of ransomware. Reference [11] built a static model called R-PackDroid that was light weight solution and was implemented on users' device itself. Its functionality was to extract and analyze the application packages from the apk files. Reference [12] extended their previous work, which has attained a considerable improvement. For the successful implementation of their designed experiment, they gained the administrative rights by rooting the device. After getting the root access, they performed extensive testing on the several applications like financial applications and social applications. Their experiments observed that most of the crucial applications do not fulfil the minimum-security requirements, which increase the chance of data leakage. Reference [13] made use of hierarchal steps of analysis before installation of an application to guarantee its trustworthiness. It has the capability of labelling each application in one of categories as either trustable or type of risk associated with it, i.e., high risk, low risk, and medium risk. For its successful implementation, its analysis is based on multiple information being gathered like permissions used by applications, number of downloads, source of the application, and its rating and developer reputation also. This

approach does not include code-based detection as it used only application metadata. They proved their approach as an effective as well as reasonable approach. Reference [14] developed a completely automatic malware identification mechanism. Its results are based on the multiple classifiers which categorize each application as benign or malicious with the appreciable accuracy of 82.93%. For their experimental observations, they used a very large set of applications containing 107,327 safe and 8,701 malicious applications along with the feature set of top 34,630 out of 23,74,340 features. To maintain balance between the performance and results of all the classifiers, they collaborate performance of all. Reference [15] used supplementary techniques that have always played important role when combined with conventional techniques. Here, in this paper also, authors have firstly captured the metadata of each application and their associated features like developer info, number of downloads, application creation date and time, and permissions being granted. Then, further it applies appropriate machine learning algorithms to assess and analyze. This is a simple and effective approach to gain high performance accuracy. The literature also suggests deep learning feature fusion for identifying mobile malware [16]. Research trends in Android literature have been performed by authors and suggest that machine learning has ability to achieve better accuracy [17, 18]. According to authors [19], healthcare organizations are the key targets of ransomware attack due to the vitality and confidentiality of patient data and then comes the governmental institutions as criminals know the importance of data for the government and they expect to get back the ransom. The third main target of ransomware attack is higher educational institutions due to weak IT hierarchy and then comes the law firms and mobile users who become the target of ransomware attack. Table 2 shows rank-wise targets organizations affected by ransomware attacks.

Reference [20] formulated a hybrid technique called as MONET which is based on the static as well as dynamic analysis. In this model, behavior of the user is consistently monitored and mapped against the run-time behavior of the malicious application. It also includes signature matching generated on the basis of API calls. The significant aim of this approach was to identify malware as well as its variants. Reference [21] attempted to provide full protection against malware, and most importantly this model gives descriptive analysis to users about the threat and its awareness measures. This model sustained high performance accuracy as it is a three-step fold mechanism. It makes use of combined benefits of multiple approaches like static and dynamic and further merged it with effects of machine learning algorithms or local-remote hosts. First, it includes static analysis using a famous framework called Drebin [22] feature set. It also then applied dynamic analysis with the use of system calls which actually improves their analysis results. Further, it applies appropriate machine learning concepts and local-remote host concepts to strengthen their performance accuracy. Reference [23] observed that library component does contain some instances of its malicious behavior. Based on this apparent observation, authors developed a unique approach in which they detect the malwares on the basis of abnormal library instances. The major part of the whole process emphasizes on to find whether a library instance has been renamed or not. For the demonstration of their framework, they used more than 1100 applications set out of which 185 were found to be malicious as their library instances were found to be abnormal. Reference [24] proposed a new framework to perform malware detection on the basis of network traffic flow. They considered all the constraints of the traditional static and dynamic techniques such as code obfuscation and resource limitation. In comparison to which, they find that their approach seems to be quiet promising. Based on the fact that most of malware develop and spread across the multiple devices during network processing, so analyzing the network flow will definitely help in identification of malicious activities associated with applications. The proposed approach performs automatic feature selection using appropriate natural language processing and achieves 99.15% detection rate. The framework was also claimed to perform better than many antivirus scanners.

Table 2: Key targets of ransomware attacks.

Rank targets	Key target organisations
1	Healthcare sector
2	Government institutions
3	Education
4	Law firms
5	Mobile and MAC users

The literature witnesses that the most of the existing frameworks consider system calls, API tracing, and static features like manifest files and permissions, for detection and analysis. On the contrary, ransomware families target the other features and also target personal information and device information. Towards the end of 2017 (Quick heal, 2018), it was reported that ransomware is making use of unique features and make frequent new variants. Examples are doubleLocker that locks both screen as well as data. Some of variants show smart behavior, and their action is based on the Internet status of the user. Such frequently emerging new features which had never been seen before pose a great challenge for existing techniques [1]. However, the engineering new feature fusion method to support in-depth study of all ransomware families is the need of the hour.

3. Methodology

This section presents the overall methodology followed for the hybrid framework to mitigate Android ransomware. We have included details of data collection, feature extraction, feature selection, and machine learning classifiers. To enhance the effectiveness of the proposed hybrid framework, we have built the feature selection algorithm to extract k-dominant features. This proposed hybrid framework also utilizes the various machine learning models to classify each apk file as ransomware or clean.

3.1. Data Collection. In this experimentation, applications are collected from two major sources. For clean applications, around 1486 apk files have been downloaded from Google Play Store. Google Play Store is an official Android market that promises to provide the most trusted source of applications. Google Play Store developer and support team claim that they do not permit applications which mine the cryptocurrencies [25]. For malicious data samples, Android Malware Dataset (AMD) is used. AMD is a standard repository which officially provides access to its dataset especially for research purpose [26] and has been used by many researchers in their study [27–29]. AMD provides updated and latest release for its collection. AMD dataset contains thousands of malicious applications. In this work, we have included only ransomware families which cover 1763 ransomware samples. Here, in this study, Android applications are termed as clean applications or ransomware applications as in Table 3.

3.2. Proposed Hybrid Framework. Figure 2 presents the overall methodology of the proposed hybrid framework. A large set of 3,249 application samples containing both clean and malicious samples are used as input. First, static analysis is performed on each application in data sample to extract static features associated with that application. Further, dynamic analysis is performed to extract dynamic features set used by both benign and ransomware applications. Static analysis and dynamic analysis are performed in parallel to extract feature set. Further, we transformed the obtained static and dynamic feature set information to build a combined feature vector matrix. A well-designed feature selection algorithm is applied to identify k-dominant features. This algorithm is applied iteratively to identify k-dominant feature where k is set to be 20, 40, 60, and 80. To evaluate the effectiveness of model, machine learning models are applied to train and classify each apk files as clean or ransomware application.

3.3. Feature Extraction. In this work, we majorly focus on manifest.xml file to extract static properties associated with that application. Manifest file provides metadata like package name, acquired permissions, and related application components, i.e., activities, broadcast receivers, and other services required as static properties only hold features being used without executing an app. For advanced cyber-attacks, it becomes important to check actual behavior analysis of application. Hence, in this work, we also performed dynamic analysis of each apk file to extract the dynamic features. To evaluate the effectiveness of static and dynamic techniques over Android ransomware applications, we analyzed a few applications statically as well dynamically. We observed the similarity in feature usage pattern among clean as well as ransomware samples. Common features are considered to be the most dangerous features. It becomes very important to scan those static and dynamic features for better results. This laid the formation of the algorithm for extracting dominant features for our proposed hybrid framework as discussed in subsequent sections.

3.3.1. Static Feature Extraction. For extraction of static properties associated with application, we have used Apk tool version 2.4.0 [30] as shown in Figure 3. Apk tool is a popular open-source tool that decompiles apk file to extract its code and other metadata details. The decompressed files contain manifest.xml, resource folder, and java code. Permissions are generally considered to be one of the most important static properties. Each application acquires a set of permissions upon installation. These permissions can be easily extracted from manifest.xml file. In this work, python scripts are used to extract permissions using Apk tool. Apk tool decompiles each apk file, extracts the permissions associated with it, and helps store the information in a text file format. The scripts involve the following steps:

(i) The script requires .apk file as input
(ii) It uses Apk tool v2.4.0 to decode .apk file to xml file, dex files, and other resource folder
(iii) The script scans manifest.xml file to extract all permissions using "permission" tag
(iv) Further, this information is stored to text file format
(v) These steps are repeated for all the .apk files

The working of script for static feature extraction using Apk tool is as shown in Figure 4. The steps are repeated for all applications, i.e., apk files in the collected dataset. As the dataset contains both clean and ransomware applications, static permission is analyzed thoroughly. Further, we observed similarity in feature usage between clean and ransomware applications. It was found that permission used by clean applications is quite similar to permissions being used by ransomware samples. Hence, those permissions are considered to be riskiest permission and must undergo checks for analyzing any application. The details of permission extracted are shown in the result section.

Apk tool takes up the largest proportion and is often used to decompile APKs. Current support tools for static analysis and its percentage of use in other studies [31] are enumerated as shown in Figure 5.

A study over Android detection mechanisms using static features also confirms that around 41% of techniques used permissions as a key parameter for detection and analysis of Android malware [31]. Other features used are API calls, metadata, intents, and so on as depicted in Figure 6.

3.3.2. Dynamic Feature Extraction. Dynamic analysis observes the actual behavior of an application while in execution. It is quite obvious that on-device real time execution of Android application on Android platform will result in high consumption of battery and other device resources. Hence, in this study, dynamic analysis is performed on the virtual emulated environment to examine its dynamic features as shown in Figure 7. The literature states that dynamic features like API calls, permissions, and system calls are frequently used features. In this work, emulator called habo analysis system [32] is used which has capability to scan and extract other set of dynamic features also. Security analyst

Table 3: Difference in clean and ransomware applications.

	Clean applications	Ransomware applications
Characteristics	These applications do not contain malicious code in the source code. These are safe for device.	These applications do contain malicious code in the source code. Malware authors, i.e., attackers may inject the code to affect the device users.
Installation	Upon installation of clean applications, it performs its dedicated task and does not harm either the device or user's data.	Ransomware applications encrypt the confidential data and file in system upon installation. These can even lock the device and demand ransom to unlock it.

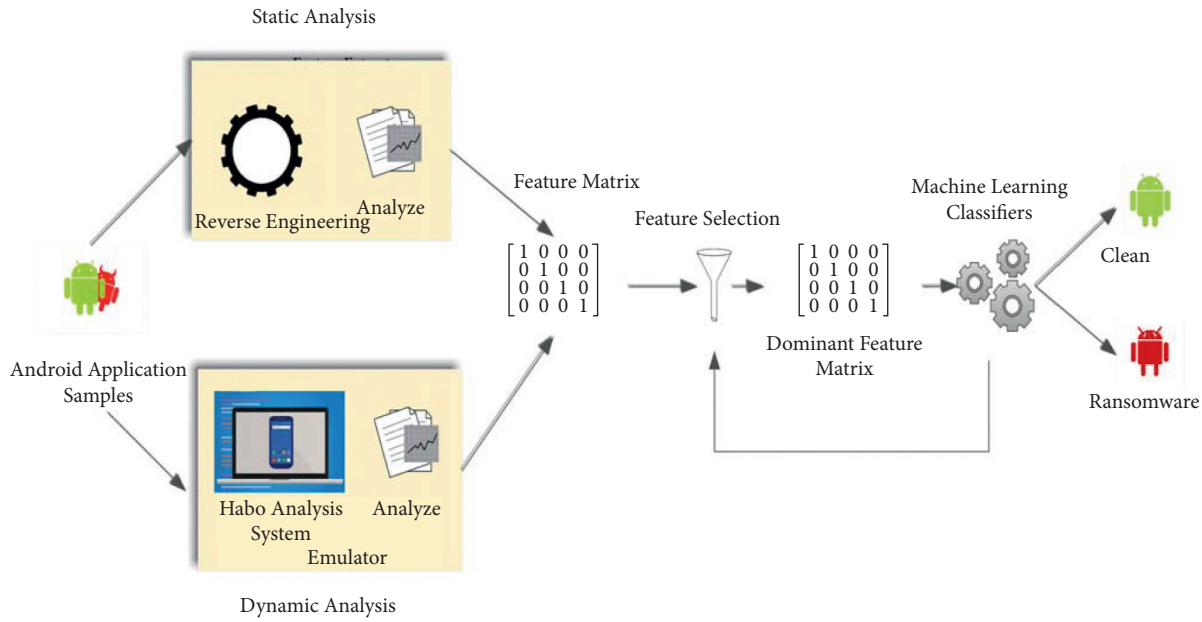


Figure 2: Proposed hybrid framework.

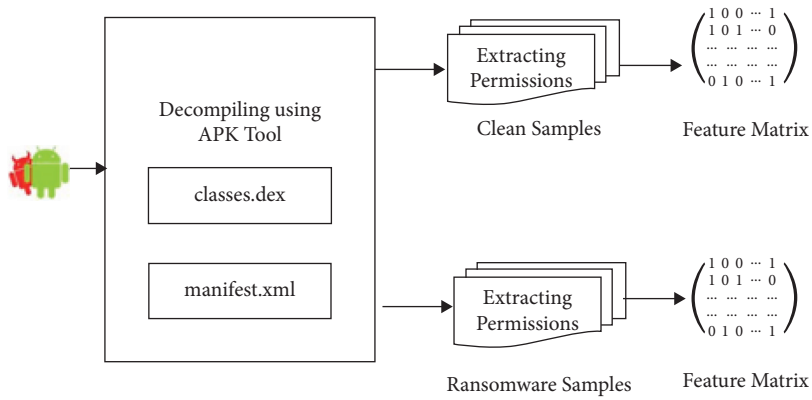


Figure 3: Static feature extraction.

generally used the habo analysis system to automate the process of malware analysis. Dynamic features used in this proposed work include API calls, permissions, system calls, network, file monitoring, and other system components. A robust approach to perform an effective dynamic analysis lies in extracting a limited set of features that provide the ability to classify between ransomware and benign behavior of application being tested. For which, we have used the prominent feature selection algorithm as discussed in subsequent sections. The traces obtained upon execution under controlled virtual environment are recorded to generate the individual reports. These reports contain significant information about dynamic features like API calls, permissions, system calls, network, file monitoring, and other system components. Further, these generated reports are converted to required input format for the experiment. The steps followed for extracting the dynamic features are as follows:

```
I: Using Apktool 2.4.0 on es-file-explorer-4-2-1-9.apk
I: Loading resource table…
I: Decoding AndroidManifest.xml with resources…
I: Loading resource table from file:_WorkArea1\Frameworks\1.apk
I: Regular manifest package…
I: Decoding file resources…
I: Decoding values/ XMLs
I: Baksmaling classes.dex…
I: Baksmaling classes3.dex…
I: Baksmaling classes2.dex…
I: Copying assets and libs…
I: Copying unknown files…
WORKING ON EXTRACTING PERMISSIONS
FINISHED.
```

Figure 4: Working of apk tool.

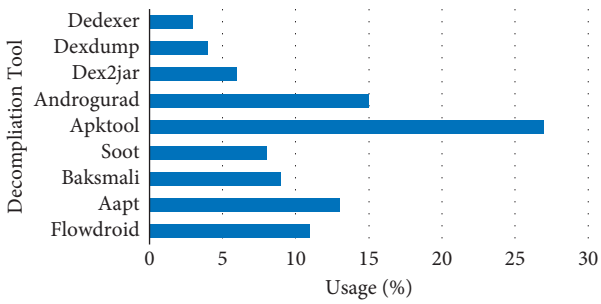


Figure 5: Use of apk tool.

(i) Set up the environment settings

(ii) Install VirtualBox 5.1

(iii) Upload the source code to virtual machine to compile it

(iv) After successful compilation, upload .apk file

(v) For each application in dataset,

  (a) Test and analyze the application

  (b) Download the output.dynamic report file

3.4. Feature Vector. In this step, the recorded features from the previous step were transformed into nominal representation to build feature vector. The feature used by application is marked as 1 denoting its presence and 0 in case of its absence.

Let us assume an application that uses set of features (f_1, f_2, …, f_n). For every application in collected dataset, i.e., clean as well as ransomware, f_n is calculated based on formula as follows:

$$f_n = \begin{cases} 1, & \text{if feature exists,} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

3.5. Feature Selection Using Prominent Feature Selection Algorithm. Static analysis is based on code and signature similarity and fails at code obfuscation. Dynamic analysis techniques are strong enough to withstand with vulnerable situations and can even detect suspicious behavior even when code is compressed or encrypted. Smart malware actions are sometimes triggered only under certain conditions, which is not possible to achieve in emulated testing environment. Hence, we have used a fusion of static features with dynamic feature to produce promising results for hybrid solution for Android ransomware. Existing hybrid solutions majorly vary in feature set used for detection of Android ransomware. Ransomware families utilized new evolving features for constructing new variants. The success of any approach directly depends on feature set and feature selection method being used.

The feature set must be unique for both clean feature vector and ransomware feature vector. Hence, a unique feature set is created by taking combination of all the static and dynamic feature sets used by clean samples and ransomware samples. Initially, extracted static features and extracted dynamic features were large in number and redundant. Further, a total of 94 features were extracted as unique set of features as shown in Table 4. Considering all the features or larger set of feature combination for analysis and classification may lead to redundant data. Moreover, to maintain the accuracy and effectiveness of results, we have used prominent feature set in this work. To identify the most significant features, we used a feature selection algorithm as stated Algorithm 1. This algorithm determines top k-dominant features being used by both ransomware and clean applications.

Feature vector files contain data in the form of zeros and ones to represent existence and absence of each feature fed as an input. Further, we calculated sum of frequencies of each feature in clean feature vector file and further normalized it by dividing it with total number of samples, i.e., for clean as well as ransomware samples. The value of nominal frequency for each feature determines its dominance. Then, we calculated the absolute difference between both normalized frequencies for each feature. It represents similarities in feature existence in both clean and ransomware samples. For extracting the most used features, we sorted all the values in ascending order. The smaller values of difference signify more dominance of that feature whereas higher the difference, lesser the dominance of the feature. Initially, we identified the top 20 most dominant features to analyze and classify the samples. However, we have also iteratively increased the number of dominant features by 20 at each step. However, it is expected that considering the large number of feature combination may result in high consumption of system resources as well as time. The difference in nominal frequencies of features among clean and ransomware applications is discussed in the result section.

3.6. Classification Using Machine Learning Models. The obtained combination of unique set of static and dynamic features is used to train machine learning models. In this work, we have used supervised learning. Two class labels used are c for clean application and r for ransomware for training the classification models. Existing solutions [33, 34] have suggested many classifiers and attained promising results. So, during our experiments, we have used multiple classifiers to test and validate our results which includes random forest [35], decision tree (J48) [36], logistic model

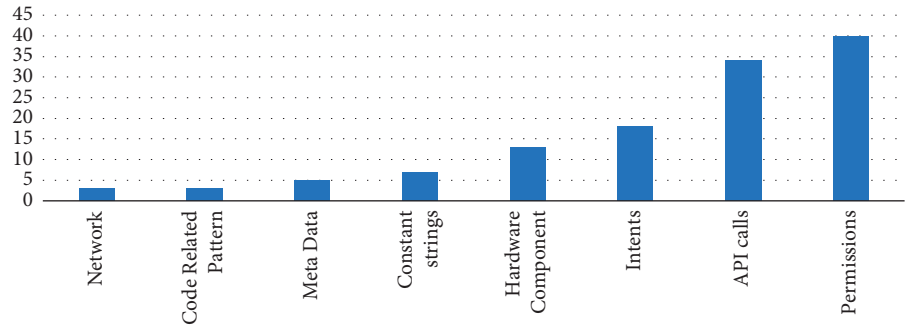


Figure 6: Use of permissions.

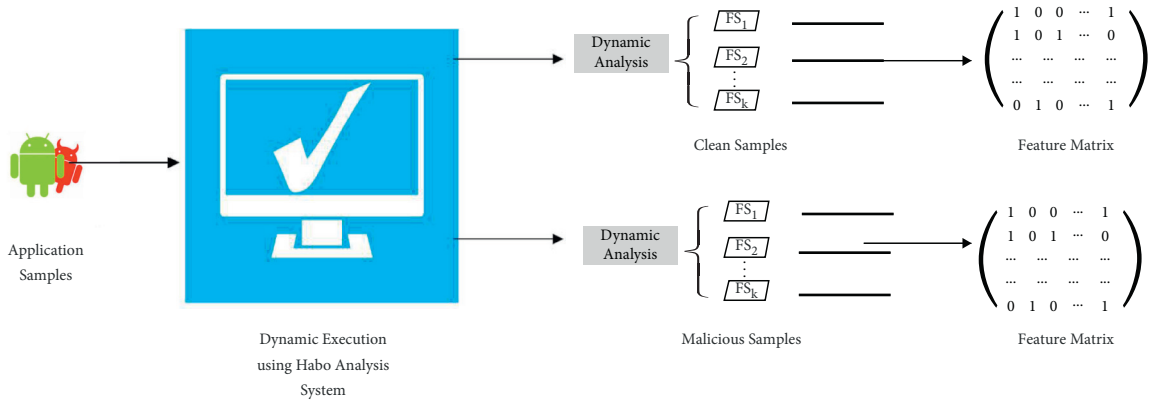


Figure 7: Dynamic feature extraction.

tree, and random tree. Selection of the correct number of dominant features was critical decision of the feature selection phase. Initially, top 20 dominant features were selected. Further, experiment was repeated by incrementing dominant features by 20 at each step till 80, i.e., 20, 40, 60, and 80.

3.7. Performance Evaluation. For measuring the performance evaluation statistics, we have used the following metrics.

3.7.1. Accuracy. Accuracy of machine learning models can be found by dividing the total number of correctly classified with sum of actual positives and actual negatives. The formula for calculating the accuracy is as follows:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} * 100. \quad (2)$$

3.7.2. Recall. Recall is fraction of true positive with sum of true positives and false negatives. The equation for calculating recall can be found as follows:

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (3)$$

3.7.3. Precision. Precision is division of true positive with sum of true positives and false positives. The equation for calculating precision can be found as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \quad (4)$$

3.7.4. F-Measure. A good score of precision and recall will lead to a good *F*-measure of the model. This value represents the harmonic mean and justifies the strength of the model for classification. The formula for its calculation is as follows:

$$F - \text{measure} = \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \times 2. \quad (5)$$

4. Experimental Results

The experimental results of this study are discussed in this section. Intense manual analysis over initially obtained feature set helped to identify the most dangerous features used by Android ransomware as discussed in Section 4.2. To determine the relevance and dominance of feature, we analyzed results with varying number of features during the feature selection algorithm as discussed in Section 4.3. Classification results with top *k*-dominant features and their corresponding performance evaluation are presented in Section 4.4.

Table 4: Unique features list extracted.

Feature_No	Feature_Name
f1	Access URL
f2	Access database
f3	Access location
f4	Access mail session
f5	Access network
f6	Access network state
f7	Access shared app data
f8	Activate device manager
f9	Active activity
f10	Active ActivityForResult
f11	Add alert window
f12	Add view
f13	Aquire root access
f14	Call setAction of intent
f15	Change WIFI (wireless fidelity) state
f16	Change component property
f17	Change network state
f18	Check available GPS
f19	Check root access
f20	Create database
f21	Create file
f22	Create new process
f23	Detect device id (antisimulator)
f24	Detect operator brand (antisimulator)
f25	Disable keyguard
f26	Execute SQL query
f27	Execute system command
f28	File read
f29	File remove
f30	Get WIFI state
f31	Get accounts
f32	Get connected WIFI
f33	Get device id
f34	Get installed app
f35	Get last location
f36	Get main intent of apk
f37	Get phone number
f38	Get running service
f39	Get running task
f40	Get scanned WIFI
f41	Get special property of simulator
f42	Get specific account
f43	Get standby state
f44	Get stored WIFI
f45	Get user id
f46	Hide from desktop
f47	Initialize URI
f48	Initialize URL
f49	Initialize intent
f50	Initialize monitor driver file
f51	Initialize new process
f52	Install shortcut
f53	Intercept broadcast
f54	Kill background processes
f55	Launch apk via intent
f56	Load class
f57	Load dynamic library
f58	Load website in webview
f59	Make toast
f60	Monitor network data
f61	Open bluetooth
f62	Parse URI
f63	Read URL data
f64	Read call log
f65	Read external storage
f66	Read history bookmarks
f67	Read one line from buffer
f68	Read system settings
f69	Receive network data
f70	Record audio or media
f71	Register receiver
f72	Reset password
f73	Run-time error
f74	Scan WIFI
f75	Send broadcast
f76	Send extra information
f77	Send mail via intent
f78	Send network data
f79	Send notification
f80	Send SMS
f81	Set looped task
f82	Set timed task
f83	Start recording
f84	Start service
f85	Stop recording
f86	Uninstall shortcut
f87	Vibrate
f88	Window information
f89	Write external storage
f90	Write file
f91	Write system settings
f92	SetSharedPreferences
f93	AddAppToShareData
f94	ReadSharedPreferences

4.1. Experimental System Setup. Being a hybrid approach, we required a good device and other computational resources for our experiments. It includes both static and dynamic analyses of a large dataset of 3249 application samples. Table 5 shows the details of system setup and tools used during the experiment.

4.2. Feature Extraction and Critical Analysis over Obtained Feature Set. All the static and dynamic execution reports were transformed to feature vector format to analyze obtained features for both clean and ransomware samples. Nominal values for each feature show whether a particular feature is used by that sample or not. Based on reports of clean and ransomware feature vector statistics, we identified top 30 features used by clean samples as well as top 30 features used by ransomware samples as shown in Figures 8 and 9 separately. The results showed that ransomware application sample uses many crucial features also, and moreover a few clean application samples are also used. It becomes cumbersome for analyst to make decisions. For example, our results show that the use of feature Access Network (f5) is 75% by clean applications whereas 82% use

Input: Unique feature vector data for both clean and ransomware samples
Output: List of k-dominant features
Symbols Used: Let S_c be the total number of clean sample, S_m be the total number of ransomware samples, and K be the number of dominant features required to be extracted
Step 1: for all clean samples, calculate sum of frequencies of each feature and normalize it $\text{Normalized\_Frequency}_{\text{Clean}}(fi) = \sum_{i=0}^{n} \text{Frequency}(fi)/S_c$
Step 2: for all ransomware samples, calculate sum of frequencies of each feature and normalize it $\text{Normalized\_Frequency}_{\text{ransomware}}(fi) = \sum_{i=0} \text{Frequency}(fi)/S_m$
Step 3: for all features in unique feature list, calculate the absolute difference between normalized frequencies of clean and ransomware sample $\text{Diff}_{\text{Normalized}_{\text{Frequency}(fi)}} = \text{Normalized}_{\text{Frequency}_{\text{Clean}}(fi)} - \text{Normalized\_Frequency}_{\text{ransomware}}(fi)$
Step 4: Sort $\text{Diff}_{\text{Normalized}_{\text{Frequency}(fi)}}$
Step 5: Choose k to record k number of dominant features for k in (20, 40, 60, 80) iteratively.

Algorithm 1: Dominant feature selection.

Table 5: Experimental system requirements.

Static analysis tool	Apk tool v2.4.0
Dynamic analysis tool	Habo analysis system
Data mining tool	Weka 3.8.3
Operating system	Windows 10
Processor	Intel(R) core (TM) i5-8250U CPU@ 1.80 GHz
RAM	8.0 B

in ransomware applications samples. Access Network indicates establishing communication with Internet which can be very dangerous in case of ransomware application. Similarly, features like Send Network Data (f78), Receive Network Data (f69), and Send Extra Info (f76) have been observed to be 10–15% more in use than a normal clean application sample. Making communication with command and control servers is the major step involved in ransomware working mechanism. So, it justifies that it is important to check such critical features while analyzing application against ransomware attacks. The use of file operations like File Read (f28) and File Remove (f29) do not differ in large, hence should be added to list of risky features. Based on the observation made, we intend to focus on similarity in feature usage pattern among clean as well as ransomware samples. Common features which do not differ in large are considered to be the most dangerous features. It becomes very important to scan those static and dynamic features for better results. To produce effective results, we have used the feature selection algorithm for extracting dominant features for our proposed hybrid framework as discussed in subsequent sections.

4.3. Feature Selection. Initially, we analyzed all the features of all the samples and found that occurrences of usage of some features in clean and ransomware applications differ in large. To record the difference in nominal frequency of each feature among clean and ransomware samples, we tend to find k-dominant features as discussed in Algorithm 1. Further, we implemented the experiment by varying the value k as 20, 40, 60, and 80. The varying k helped to perform cross-analysis about dominant features over all the collected

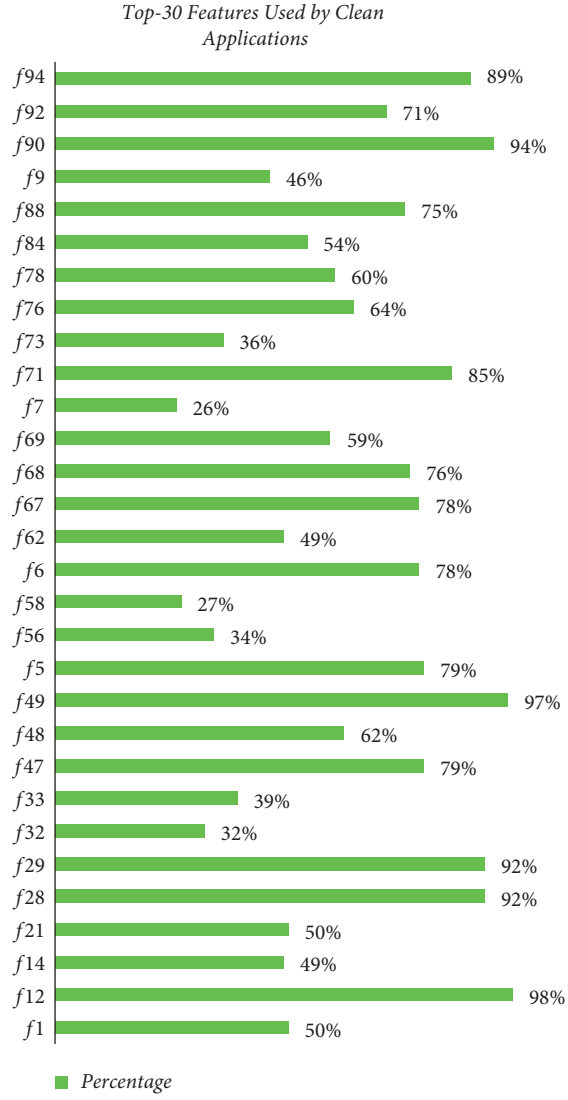


Figure 8: Top 30 features used by clean samples.

samples. The dominant features distinguish the differences in the behavior of clean and Android ransomware applications. Here, graphs as in Figures 10–15 represent

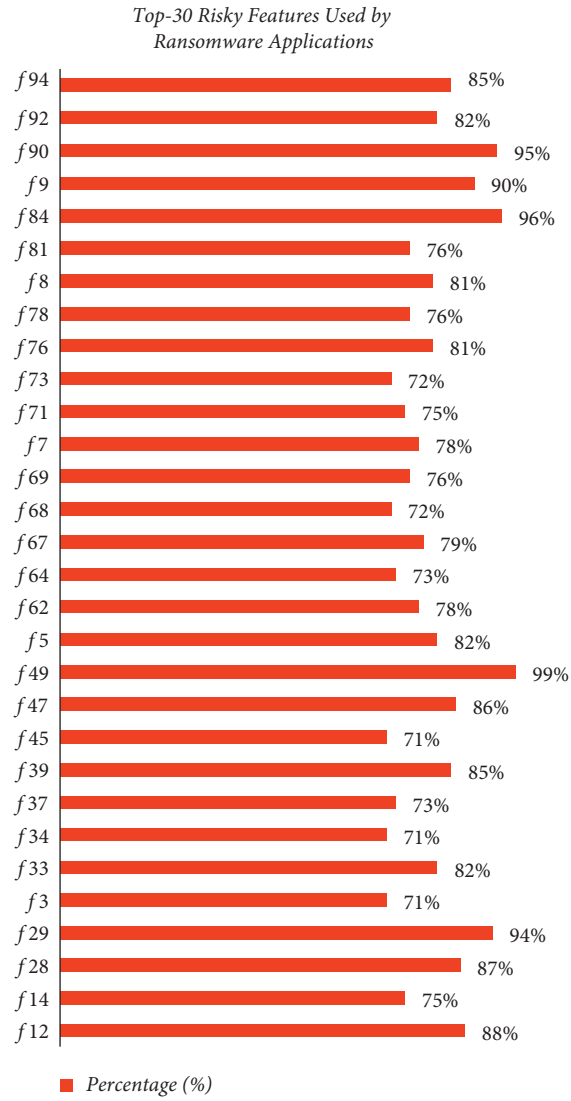


Figure 9: Top 30 risky features used by ransomware samples.

difference of normalized feature occurrence for top 20, 40, 60, and 80 dominant features, respectively.

The results of top 20 dominant features extracted include Access URL (f1), Access location (f3), Access shared app data (f7), Activate device manager (f8), Active Activity (f9), Create file (f21), Get device id (f33), Get installed app (f34), Get phone number (f37), Get running service (f39), Get user id (f45), Initialize URL (f48), Load class (f56), Load website in webview (f58), Read call log (f68), Run-time error (f73), looped task (f81), Set timed task (f82), Start service (f84), and Window information (f88). We observed that clean applications generally do not use much of a few features like Activate device manager (f8), Read call log (f68), and Get running service (f39) but ransomware applications do.

However, top 40 dominant features include all the features extracted as top 20 list as well as a few more features like Access Database (f2), Access Network State (f6), Call setAction of intent (f14), and Check root access (f19). Features like Access Database (f2), Access Network State (f6), Call setAction of intent (f14), and Check root access (f19) are majorly used by ransomware applications to perform kernel level check to attain the root access and device admin privileges. The results also justify that our feature selection algorithm is able to identify the most crucial features which must be included for analysis procedure.

Similarly, we have also identified top 60 and top 80 dominant feature lists for our experiments. Selection of the correct number of dominant features was critical decision of the feature selection phase. Initially, top 20 dominant features were selected. Further, experiment was repeated by incrementing dominant features by 20 at each step till 80, i.e., 20, 40, 60, and 80. To compute the effectiveness of the model, we have applied the classification model iteratively for all top extracted features as discussed in Section 4.4.

4.4. Classification. In this phase, we performed the classification over collected data set containing both ransomware and clean applications. The major purpose is to identify suitable classifier with the appropriate number of features

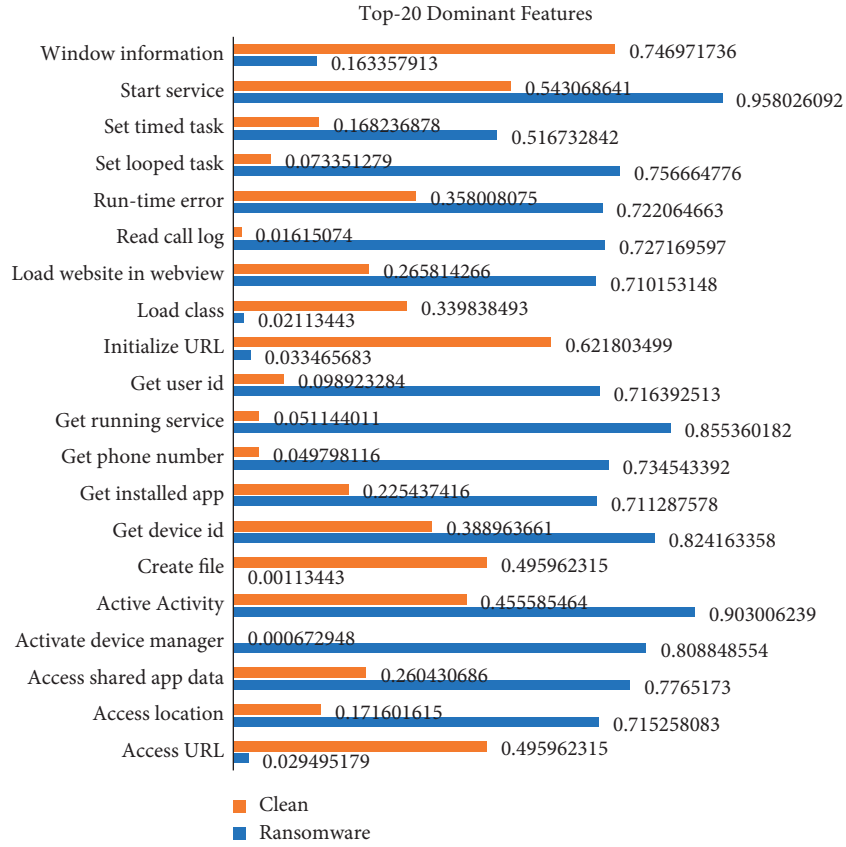


Figure 10: Difference in nominal frequencies of top 20 dominant features.

which can classify applications with highest accuracy. We have used multiple classifiers to test and validate our results which include random forest, decision tree (J48), logistic regression, and random tree. During the feature selection algorithm, we decided to extract top k-dominant features with varying value of k to be 20, 40, 60, and 80. Classification results with all values of k are presented in subsequent sections. We evaluated and compared the performance of classifiers with other performance measurement statistics, i.e., accuracy, recall, precision, and F-measure as shown in Figures 16–19.

Figure 20 shows that initially J48 produced highest false positives. Further, with the increase in the number of features set, the considerable dip represents a slight better performance than LMT (logistic model tree) and random tree. Overall, random forest produces minimal values for false positive over the change of the number of features and least when 60 dominant features were considered.

Figure 16 shows that initially random forest, random tree, and LMT produce almost the same values for false negatives. Further, with the increase in the number of features set, the downfall represents a slight better performance. However, J48 produced highest false negatives throughout different sets of dominant features. With k to be 40, random forest produced minimal values for false negatives. The rest gradually becomes stable with varying number of features.

Accuracy of any machine learning classifiers can be calculated by dividing the total number of correctly classified with sum of actual positives and actual negatives. The line chart as shown in Figure 17 illustrates that with the increase in the number of features, there is a substantial increase in performance of all the classifiers. Overall, random forest found to be the best in classifying applications sample into clean or ransomware.

The results show that among multiple classifiers, the random forest algorithm outperforms in terms of highest accuracy, lowest false negative, and false positive for all sets of features taken to be as 20, 40, 60, and 80. With 60 dominant features, random forest algorithms achieved the highest accuracy of about 99.85% with zero false negative. As the random forest algorithm is based on ensemble learning, the problem of overfitting and missing data is reduced. Due to its abundance qualities, it has also been used to detect ransomware by other researchers as the only classifier used in their studies [37, 38]. Researchers also do compare the performance of multiple classifiers, and their results also indicate that random forest performs better than random tree or any other single decision model tree [39].

4.4.1. Classification Results with Top 20 Dominant Features. Figure 18 shows effect of selecting 20 dominant features as an input dataset on F-measure along with the results obtained from computing precision and recall for multiple

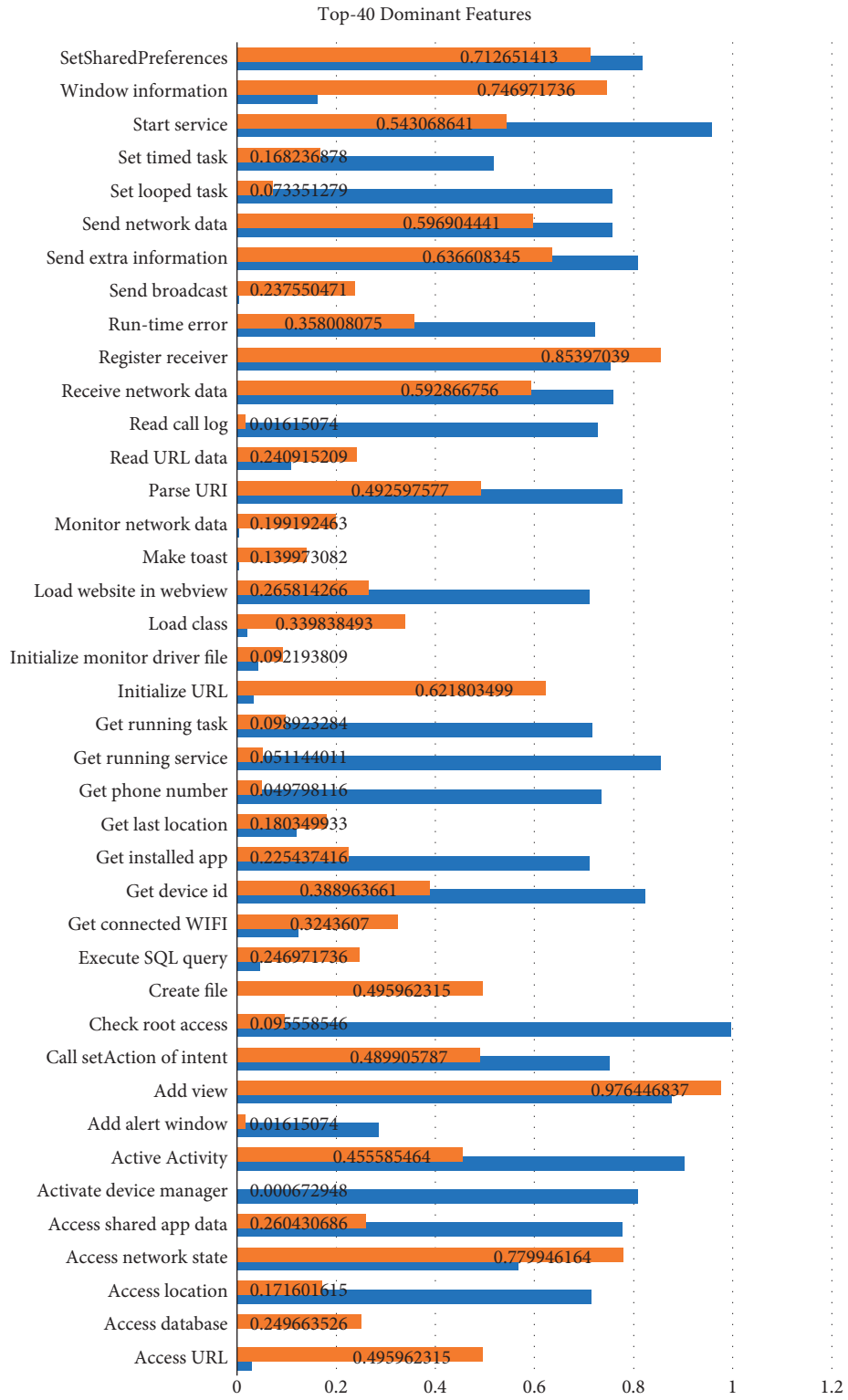


Figure 11: Difference in nominal frequencies of top 40 dominant features.

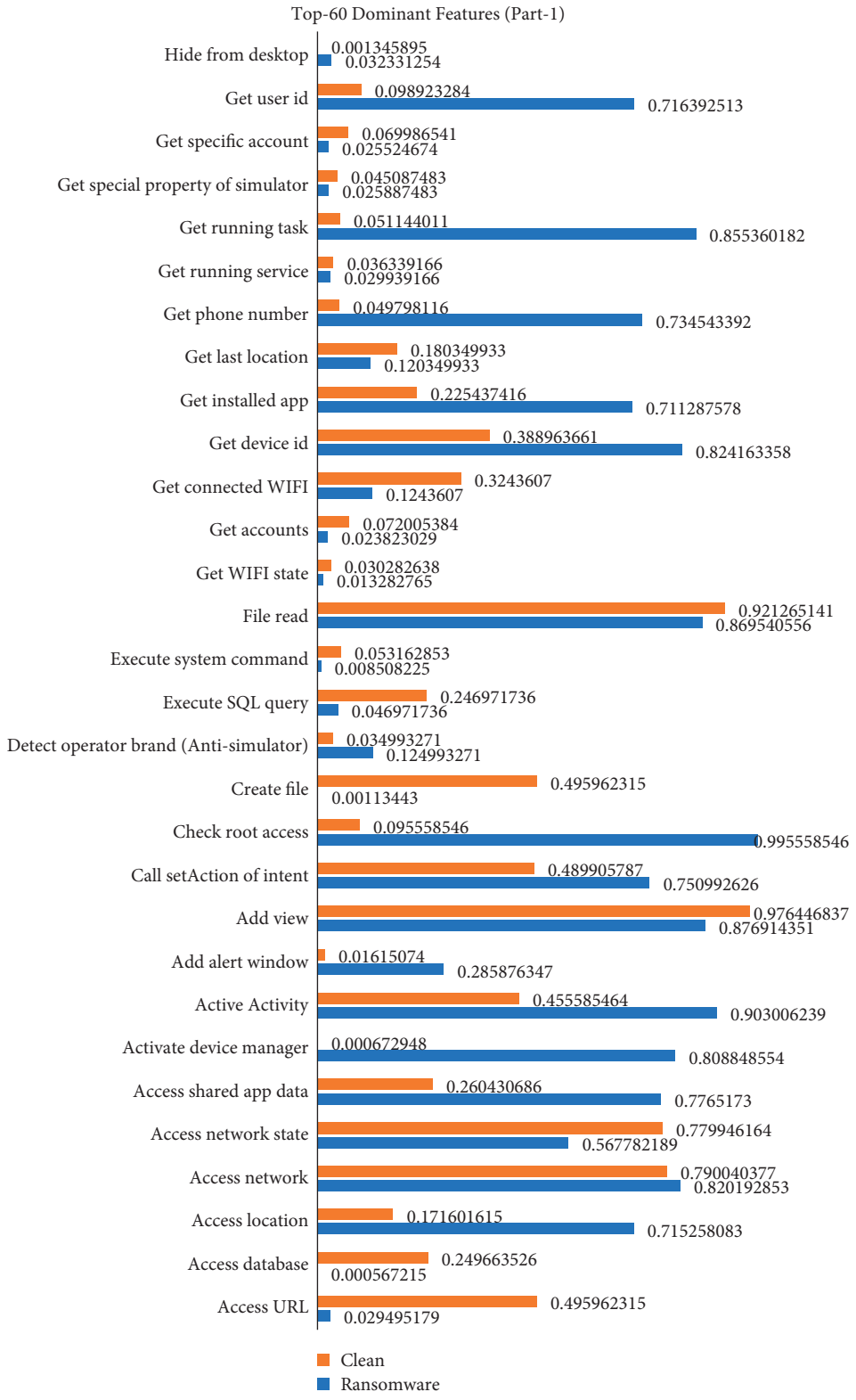


Figure 12: Difference in nominal frequencies of top 60 dominant features (Part 1).

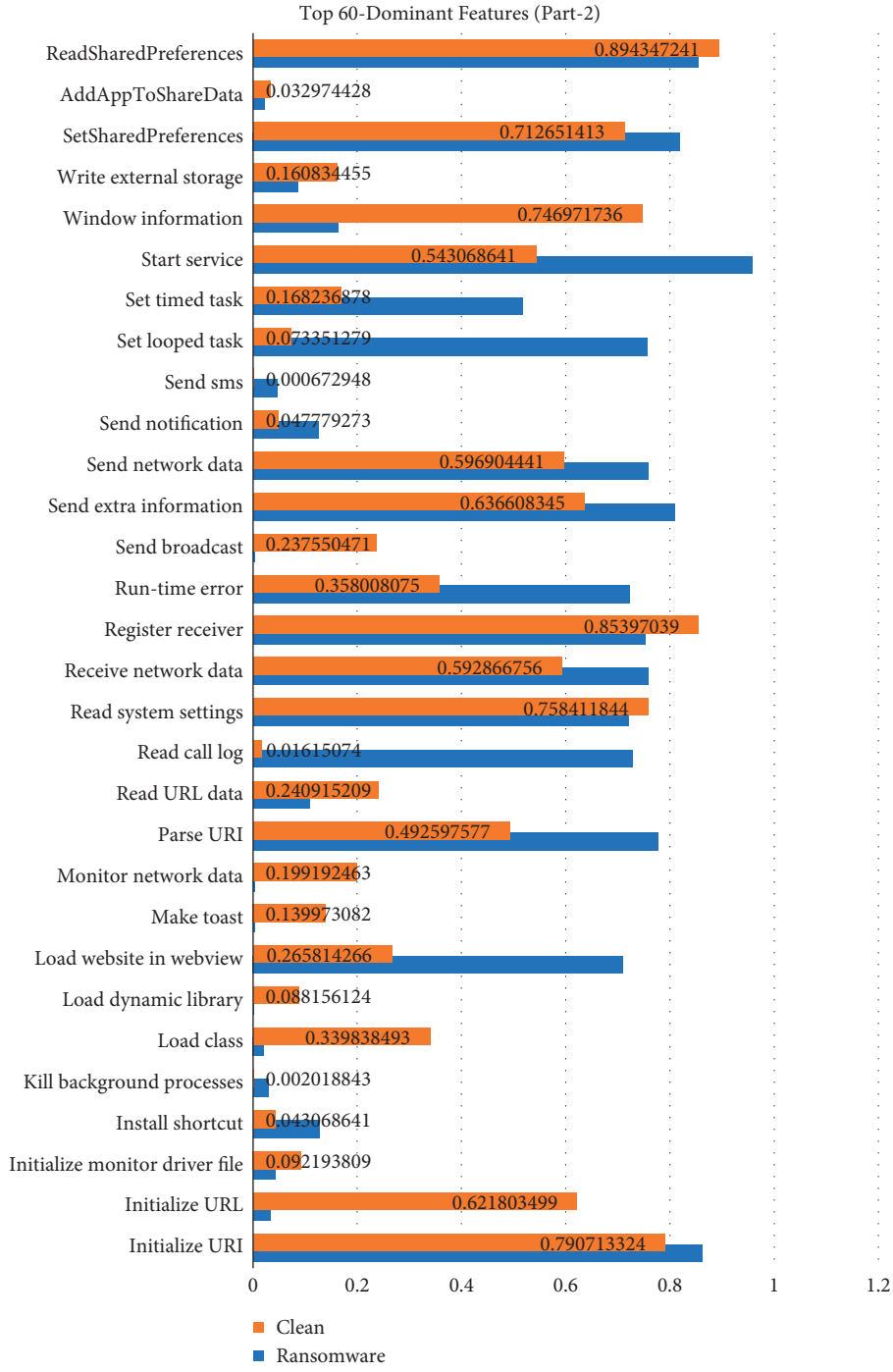


Figure 13: Difference in nominal frequencies of top 60 dominant features (Part 2).

classifiers, i.e., J48, random forest, LMT, and random tree. Computational values of random forest, random tree, and LMT are closely equivalent to each other but random forest has achieved best values of recall, precision, and *F*-measure, i.e., 0.984118, 0.986356, and 0.985236, respectively.

4.4.2. Classification Results with Top 40 Dominant Features. Figure 19 shows effect to cater 40 dominant features as an input dataset on *F*-measure, precision, and recall for multiple classifiers, i.e., J48, random forest, LMT, and random tree. Computational values of precision for random forest

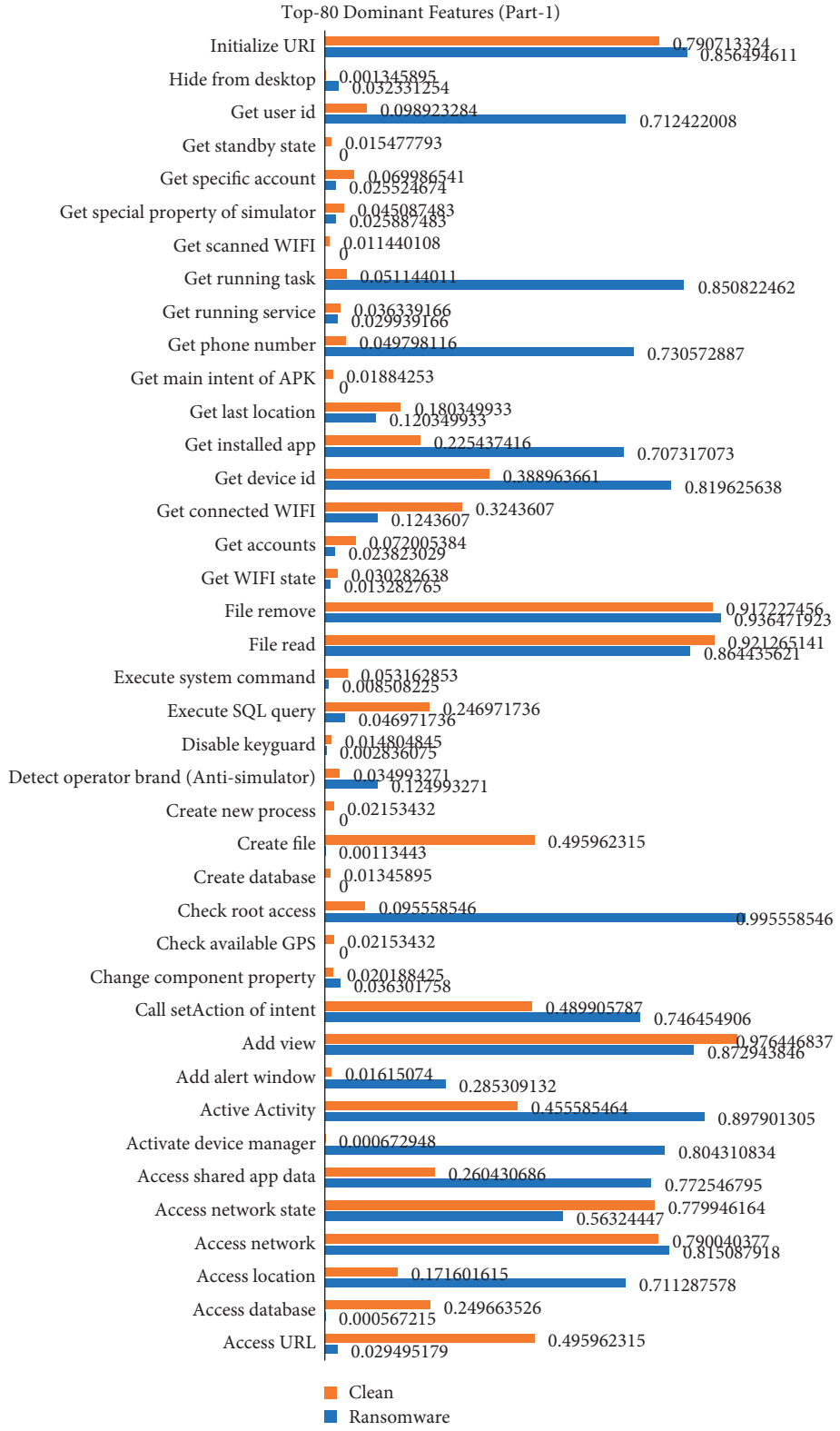


Figure 14: Difference in nominal frequencies of top 80 dominant features (Part 1).

and J48 are slightly different to each other but overall random forest has achieved the best values of recall, precision, and *F*-measure, i.e., 0.997731, 0.991545, and 0.994628, respectively.

4.4.3. Classification Results with Top 60 Dominant Features. There is dramatic change in results of 60 dominant features. Here, all the classifiers performed significantly well to classify the application samples. Figure 21 shows level up of

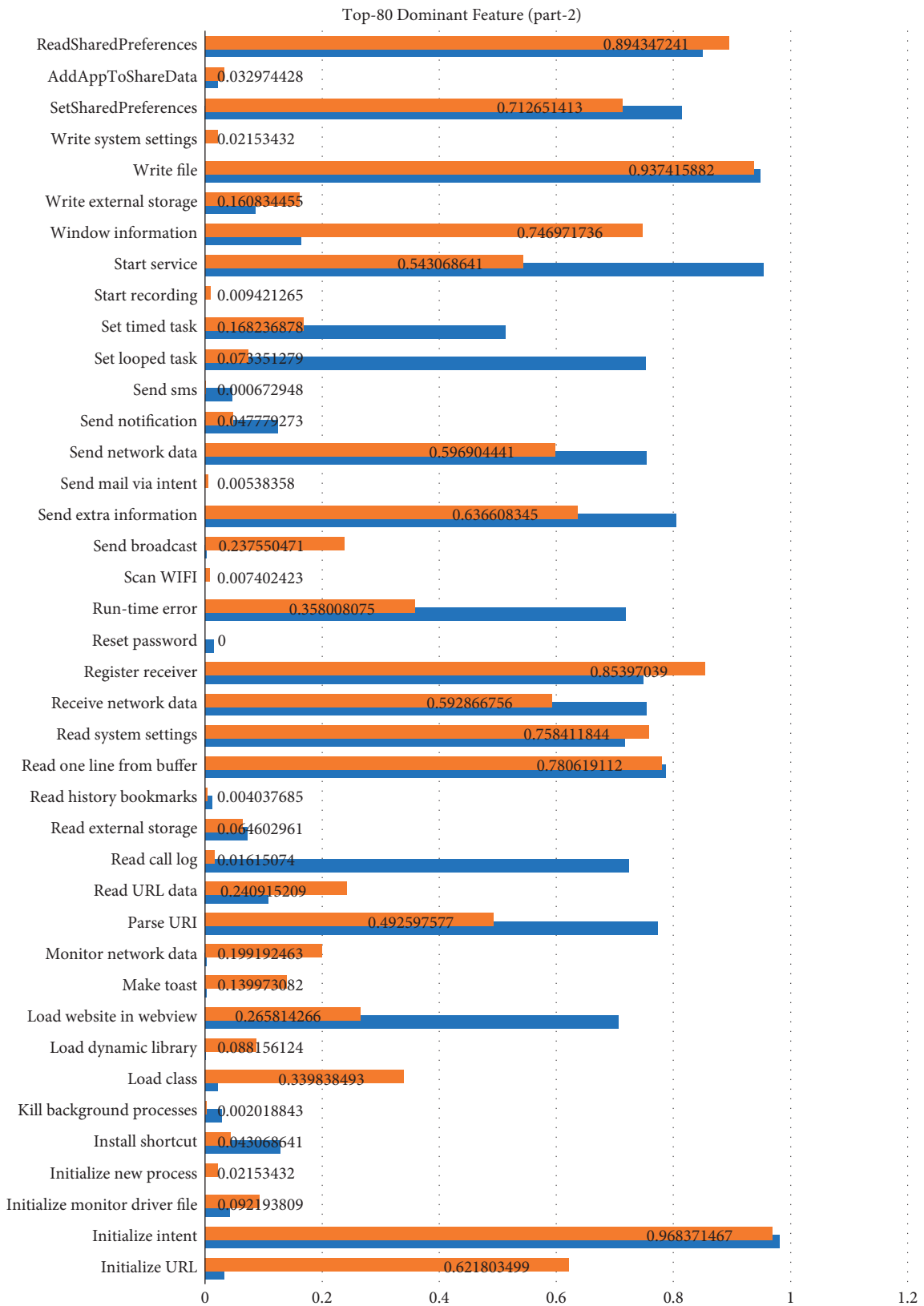


Figure 15: Difference in nominal frequencies of top 80 dominant features (Part 2).

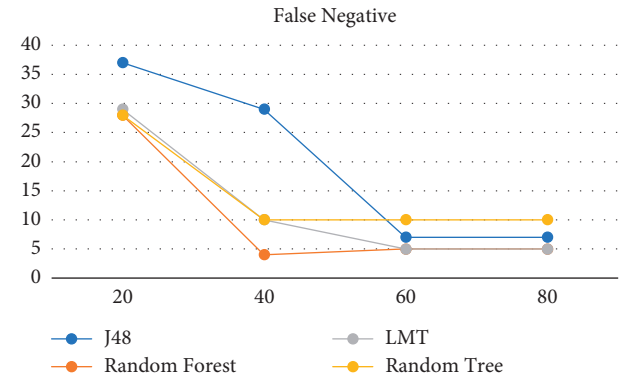


Figure 16: False negative rate.

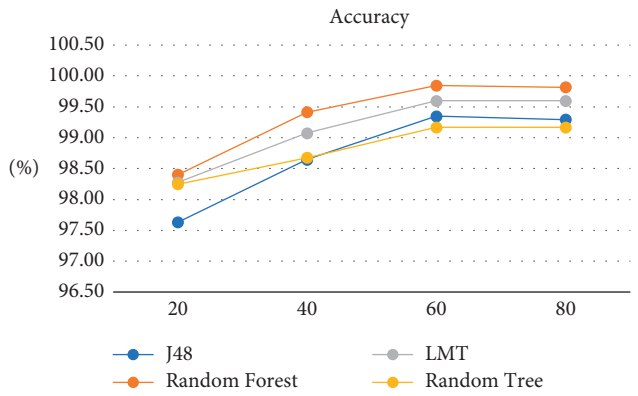
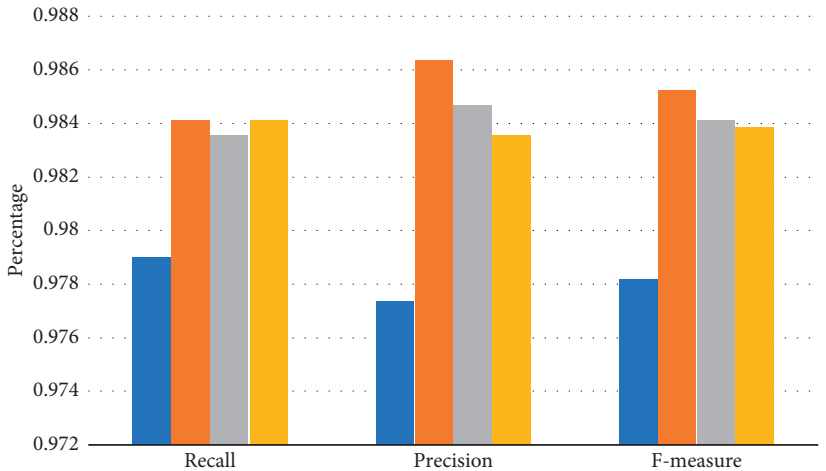
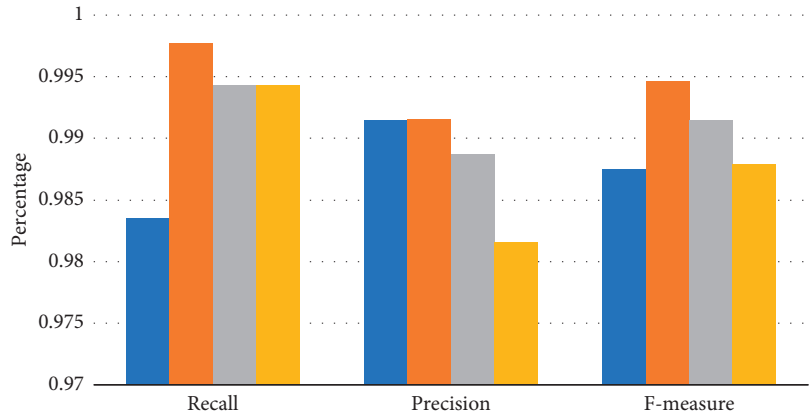


Figure 17: Accuracy of multiple classifiers.



J48	0.979013046	0.977349943	0.978180788
Random Forest	0.984117981	0.986355884	0.985235662
LMT	0.983550766	0.984667802	0.984108967
Random Tree	0.984117981	0.983560091	0.983838957

Figure 18: Classification results for top 20 features.



	Recall	Precision	F-measure
J48	0.983550766	0.991423671	0.987471526
Random Forest	0.99773114	0.991544532	0.994628216
LMT	0.99432785	0.988719684	0.991515837
Random Tree	0.99432785	0.981522956	0.987883911

Figure 19: Classification results for top 40 features.

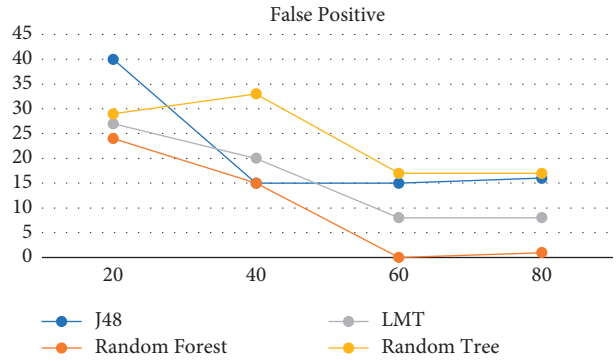
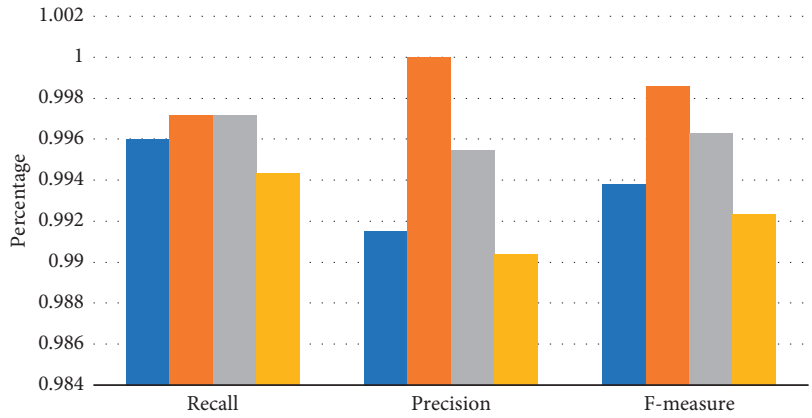
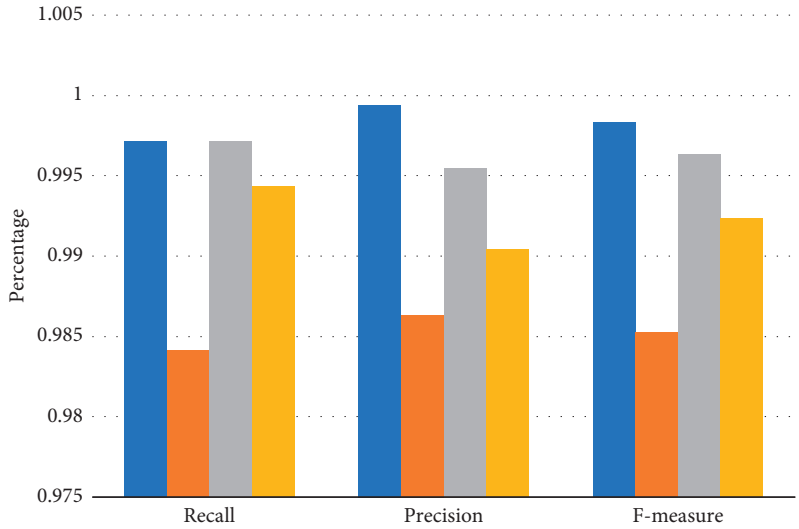


Figure 20: False positive rate.



	Recall	Precision	F-measure
J48	0.996029495	0.991530209	0.993774759
Random Forest	0.997163925	1	0.998579949
LMT	0.997163925	0.995469989	0.996316237
Random Tree	0.99432785	0.99039548	0.99235777

Figure 21: Classification results for top 60 features.



	Recall	Precision	F-measure
J48	0.997163925	0.999431495	0.998296422
Random Forest	0.984117981	0.986355884	0.985235662
LMT	0.997163925	0.995469989	0.996316237
Random Tree	0.99432785	0.99039548	0.99235777

Figure 22: Classification results for top 80 features.

random forest classifier with precision value to 1. However, results also showed that recall value of random forest and LMT is exactly same, i.e., 0.997163.

4.4.4. Classification Results with Top 80 Dominant Features. A radical change in evaluation metric results is observed on considering 80 dominant features as shown in Figure 22. There is sudden rise in performance of J48; this is due to fact that J48 performs well when there are large numbers of features. Also, recall of J48 and LMT are found to be equivalent, i.e., 0.997163.

5. Conclusion and Future Work

Existing hybrid solutions majorly vary in feature set used for detection of Android ransomware. Most of the hybrid approaches focus on a specific ransomware family or a specific ransomware type or specific feature only. Those type-specific or family-specific solutions would be difficult to consider as a generalized solution. Extracting prominent features and feature selection methods is a research challenge. We used a total of 3249 applications samples to extract the static as well as dynamic features. The experimental results show that our proposed model is able to differentiate between clean and ransomware with improved precision. The results of our proposed hybrid framework outperform the existing static, dynamic, and hybrid approaches. Moreover, it also shows that the conglomeration of all dynamic features helps distinguish ransomware more effectively. Our proposed hybrid solution confirms accuracy of 99.85% with zero false positives while considering 60 prominent features. Further, it also justifies the feature selection algorithm used.

The considerable improvement in accuracy of our proposed hybrid framework encourages the use of the novel feature selection algorithm with ensemble machine learning classifiers also. We can also demonstrate the results over a larger dataset. In future, we may train ensemble learning models to detect as well as classify the ransomware into their families. Static features like URL, signatures, strings, and other resources and dynamic features like CPU usage and time could also help in achievement of promising results. Hence, we strongly recommend the use more static and dynamic features in future.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

The presented work is PhD work of the author Tanya Gera.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: a survey and research directions," *Computers & Security*, vol. 74, pp. 144–166, 2018.

[2] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for

android," *Pervasive and Mobile Computing*, vol. 32, pp. 15–25, 2016.

[3] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *Proceedings of the 36th Int Conf Distrib Comput Syst.*, pp. 303–312, Nara, Japan, June 2016.

[4] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, "Ransomware steals your phone. formal methods rescue it," *Formal Techniques for Distributed Objects, Components, and Systems*, DisCoTec 2016, Heraklion, Crete, Greece, pp. 212–221, 2016.

[5] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 2, pp. 59–67, 2016.

[6] P. Wang and Y.-S. Wang, "Malware behavioural detection and vaccine development by using a support vector model classifier," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 1012–1026, 2015.

[7] P. Zhang and Y. Tan, "Hybrid concentration based feature extraction approach for malware detection," in *Proceedings of the 28th IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 140–145, Halifax, NS, USA, May 2015.

[8] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2015.

[9] G. Kapse, "Detection of malware on android based on application features," *International Journal of Computer Science and Information Technology*, vol. 6, no. 4, pp. 3561–3564, 2015.

[10] N. Andronio, S. Zanero, and F. Maggi, "Heldroid: dissecting and detecting mobile ransomware," in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, pp. 382–404, Berlin, Heidelberg, October 2015.

[11] D. Maiorca, F. Mercaldo, G. Giacinto, C. A. Visaggio, and F. Martinelli, "R-PackDroid: API package-based characterization and detection of mobile ransomware," in *Proceedings of the symposium on applied computing*, pp. 1718–1723, Marrakech, Morocco, March 2017.

[12] L. Casati and A. Visconti, "The dangers of rooting: data leakage detection in android applications," *Mobile Information Systems*, vol. 2018, Article ID 6020461, 9 pages, 2018.

[13] G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra, "Risk analysis of android applications: a user-centric solution," *Future Generation Computer Systems*, vol. 80, pp. 505–518, 2018.

[14] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Generation Computer Systems*, vol. 78, pp. 987–994, 2018.

[15] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "CANDYMAN: classifying android malware families by modelling dynamic traces with Markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, 2018.

[16] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of android malware images," *Sensors*, vol. 20, no. 24, p. 7013, 2020.

[17] T. Gera, J. Singh, D. Thakur, and P. Faruki, "A semi-automated approach for identification of trends in android ransomware literature," in *Proceedings of the International Conference on Machine Learning for Networking*, pp. 265–283, Springer, Paris, France, November 2020.

[18] J. Singh, T. Gera, F. Ali, D. Thakur, K. Singh, and K.-s. Kwak, "Understanding research trends in android malware research using information modelling techniques," *Computers, Materials & Continua*, vol. 66, no. 3, pp. 2655–2670, 2021, Available from: http://www.techscience.com/cmc/v66n3/41100.

[19] M. Humayun, N. Jhanjhi, A. Alsayat, and V. Ponnusamy, "Internet of things and ransomware: evolution, mitigation and prevention," *Egyptian Informatics Journal*, vol. 22, no. 1, pp. 105–117, 2021.

[20] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: a user-oriented behavior-based malware variants detection system for android," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1103–1112, 2017.

[21] 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.

[22] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: effective and explainable detection of android malware in your pocket," *Proceedings 2014 Network and Distributed System Security Symposium*, vol. 2014, 2014 Available from, Article ID 23247.

[23] H. Han, "Identify and inspect libraries in android applications," *Wirel Pers Commun*, Springer, US, pp. 1–13, 2018.

[24] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, 2018.

[25] Developer Program Policy, Last accessed on 16 December. Available from: https://support.google.com/googleplay/android-developer/answer/10286120?hl=en%0A, 2020.

[26] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *Proceedings of the Detect Intrusions Malware, Vulnerability Assess 14th Int Conf DIMVA 2017*, pp. 252–276, Bonn, Ger, July 2017.

[27] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on dex file section features," *IEEE Access*, vol. 8, pp. 10614–10627, 2020.

[28] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi, "Android malware detection based on factorization machine," *IEEE Access*, vol. 7, pp. 184008–184019, 2019.

[29] S. Turker and A. B. Can, "AndMFC: android malware family classification framework," in *Proceedings of the 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, pp. 1–6, Istanbul, Turkey, September 2019.

[30] R. Winsniewski, "Apktool: a tool for reverse engineering android apk files," 2012, http://ibotpeaches.github.io/Apktool/.

[31] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A systematic literature review of android malware detection using static analysis," *IEEE Access*, vol. 8, pp. 116363–116379, 2020.

[32] HaboMalHunter, Last accessed on 10 April. Available from: https://github.com/Tencent/%20HaboMalHunter, 2019.

[33] A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, "Extinguishing ransomware-a hybrid approach to android ransomware detection," in *Proceedings of the International Symposium on Foundations and Practice of Security*, pp. 242–258, Springer, Montreal, QC, Canada, October 2017.

[34] A. Gharib and A. Ghorbani, "Dna-droid: a real-time android ransomware detection framework," in *Proceedings of the*

International Conference on Network and System Security, Helsinki, Finland, September 2017.

[35] F. Livingston, "Implementation of Breiman's random forest machine learning algorithm," *Mach Learn J Pap*, vol. 2005, pp. 1–13, 2005.

[36] J. R. Quinlan, *C4. 5: Programs for Machine Learning*, Elsevier, Amsterdam, Netherlands, 2014.

[37] A. Altaher, "An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features," *Neural Computing & Applications*, vol. 28, no. 12, pp. 4147–4157, 2016.

[38] M. Pal, "Random forest classifier for remote sensing classification," *International Journal of Remote Sensing*, vol. 26, no. 1, pp. 217–222, 2005.

[39] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, "Random forests and decision trees," *Int J Comput Sci Issues*, vol. 9, no. 5, p. 272, 2012.