




FSDroid:- A feature selection technique to detect malware from Android using Machine Learning Techniques

FSDroid

Arvind Mahindru^{1,2}  · A.L. Sangal²

Received: 4 September 2019 / Revised: 20 August 2020 / Accepted: 22 December 2020 /
Published online: 14 January 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC part of Springer Nature 2021

Abstract

With the recognition of free apps, Android has become the most widely used smartphone operating system these days and it naturally invited cyber-criminals to build malware-infected apps that can steal vital information from these devices. The most critical problem is to detect malware-infected apps and keep them out of Google play store. The vulnerability lies in the underlying permission model of Android apps. Consequently, it has become the responsibility of the app developers to precisely specify the permissions which are going to be demanded by the apps during their installation and execution time. In this study, we examine the permission-induced risk which begins by giving unnecessary permissions to these Android apps. The experimental work done in this research paper includes the development of an effective malware detection system which helps to determine and investigate the detective influence of numerous well-known and broadly used set of features for malware detection. To select best features from our collected features data set we implement ten distinct feature selection approaches. Further, we developed the malware detection model by utilizing LSSVM (Least Square Support Vector Machine) learning approach connected through three distinct kernel functions i.e., linear, radial basis and polynomial. Experiments were performed by using 2,00,000 distinct Android apps. Empirical result reveals that the model build by utilizing LSSVM with RBF (i.e., radial basis kernel function) named as FSDroid is able to detect 98.8% of malware when compared to distinct anti-virus scanners and also achieved 3% higher detection rate when compared to different frameworks or approaches proposed in the literature.

Keywords Cyber-security · Machine learning · Dynamic-analysis · Feature selection · Permissions based analysis · Intrusion-detection

✉ Arvind Mahindru
er.arvindmahindru@gmail.com

¹ Department of Computer Science and Applications, D.A.V. University, Sarmastpur 144012, Jalandhar, India

² Department of Computer Science and Engineering, Dr. B.R. Ambedkar National Institute of Technology, Jalandhar 144011, India

1 Introduction

Today, smartphone is not only a cellular telephone, but it can integrate with the computer-like operating system, which is also able to perform various tasks with the help of apps. Symbian was the first modern mobile operating system for smartphones that entered the market in the year 2000. After that, limited mobile phone companies, like Nokia, Microsoft, Apple, and Google, has followed them and launched their own mobile operating systems in the market. Among these, Android operating system¹ launched by Google in the year 2008 is quite popular as it is freely available, open source, and has a wide range of free apps in its play store. According to Stat Counter,² Android covers 74.92% share in the market till date. However, the success of Android in the market is mainly due to its apps. Currently, about 2.6 million apps are present in the official play store of Android,³ which users can download and install for various purposes.

Android is a privilege-separated operating system where every app has its own individual system identity, i.e., Group-ID and Linux user-ID.⁴ Each app of Android runs in a procedure sandbox and accesses the permissions to use the resources which are not present in its sandbox. Depending upon the sensitivity of permissions, the system automatically grant permissions or may prompt the users to approve or reject the requests for permissions. By taking the advantage of these permissions cyber-criminals target the user privacy. As stated in,⁵ G-Data Security expert counted 3,246,284 malware apps until the end of the year 2018 and discovered over 7,50,000 new malware apps at the end of 2019. To defend Google play store⁶ from malware apps, Google introduced Google Bouncer in the year 2012, which scans new apps at the time of their launch. However, it has limitations, e.g., Bouncer can easily fingerprint.⁷ It is not hard to bypass Google's security check, so that malicious Android apps can make their way to Google Play store⁸ and ultimately to users' devices. By taking advantage of these permissions, cyber-criminal build malware apps on a daily basis and invite users to install these apps. More than two billion active Android devices are present in the market.⁹ To overcome the drawback of the bouncer and to protect Android devices, Google introduced Google play protect in the market.¹⁰ It has the capability to scan the apps in real-time. But it also have the limitations as stated in [28].

Android apps work on the principle of permission-model [11]. In addition to that, it provides protection at four level, that categorize permissions as¹¹ "signature", "signature or system", "normal" and "dangerous". In our study, we do not consider "signature" and "signature or system" because they are system granted. We only consider "normal" and "dangerous" permissions which are granted by the user. Normal permissions does not pay any risk to the user's privacy. If the permission is listed in its manifest, then it is granted by the system automatically. On the other hand, dangerous permissions give access to the

¹[https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))

²<http://gs.statcounter.com/os-market-share/mobile/worldwide>

³ <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>

⁴<https://source.android.com/security/overview/kernel-security>

⁵<https://www.gdatasoftware.com/news/2017/04/29715-350-new-android-malware-apps-every-hour>

⁶<https://play.google.com/store?hl=en-IN>

⁷<http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/>

⁸<http://gs.statcounter.com/os-market-share/mobile/worldwide>

⁹https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf

¹⁰https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf

¹¹<https://developer.android.com/training/permissions/requesting.html>

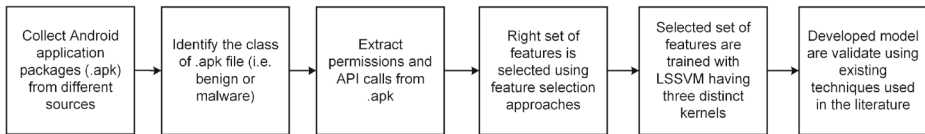


Fig. 1 Systematic approach of our proposed work

user's confidential data. However, it is purely dependent upon the user to give access or revoke the use of permission or set of permissions.

Performance of malware detection rely on selecting the right set of features. The features which are selected as an input to detect malware have a great effect on the performance of the malware detection. To select appropriate features or feature sets, in this study we use distinct feature selection approaches. Feature selection approaches are divided into two distinct classes, one class contains feature ranking approaches, and second class contains feature subset selection approaches. Feature ranking approaches are based on conclusive factors to arrange each feature according to its rank, further high ranking features are selected for a specified work. On the other side feature subset selection approaches depend upon the principle of selection of features' subset, which collectively improve detection capability. In this work, six distinct kinds of feature ranking and four distinct kinds of feature subset selection approaches are used to select the right features sets. Further, selected feature sets helps in minimizing the value of misclassification errors since it eliminates irrelevant features and holds only those features which have excellent discriminative power.

In the literature, a number of researchers applied different machine learning algorithms to detect malware. Some of the broadly used algorithms are decision tree learning algorithms [60], neural networks [54, 59], clustering [14], regression and classification. The construction of an appropriate malware detection model which can help to detect apps that are really infected with malware is still a challenging task in the field of cyber-security. So in this work, we implement LSSVM using three distinct kernel functions viz., polynomial, linear, and RBF to build a model for malware detection. LSSVM is the variant of the SVM that is established on the hypothesis of statistical learning.

The list of phases pursued by us in building an effective Android malware detection model is demonstrated in Fig. 1. To conduct an empirical study on the collection of large data set, we collect Android application packages (.apk) files from distinct promised repositories. After that, we divide collected .apk files into two classes, i.e., benign and malware based on the results of different anti-virus scanners (Microsoft window defender¹² and VirusTotal¹³). In the next phase, we extracted permissions and API calls (consider as features in our work) by using distinct tools available publicly. Additionally, the right set of features is selected by implementing feature selection approaches on our collected data set. Further, the selected feature sets (i.e., permissions and API calls) are used as an input to form a model by considering LSSVM having three distinct kernel functions. At last, to validate that our proposed model is capable to detect malware or not we validate it with some existing frameworks or approaches developed in the literature and also compared our proposed framework with different anti-virus scanners available in the market.

The novel and unique assistance of this research work is presented as follows:

¹²<https://www.microsoft.com/en-in/windows/comprehensive-security>

¹³<https://www.virustotal.com/>

- To the best of our knowledge, this is the first work in which 2,00,000 unique apps are collected that further belongs to thirty different categories of Android apps. To build effective and efficient malware detection model we extract permissions and API calls and consider them as features in this research paper.
- In this research paper, we proposed a new framework that works on the principle of machine learning algorithm by selecting relevant features using feature selection approaches. Empirical result reveals that our proposed framework is able to detect 98.75% unknown malware from real-world apps.
- Our proposed framework is capable to detect malware from real-world apps in less time-period when compared to distinct anti-virus scanners available in the market.
- Our proposed framework is able to detect 3% higher detection rate when compared to different frameworks or approaches proposed in the literature.
- In this study, we applied *t*-test analysis to investigate that features selected by feature selection approaches are having significant difference or not.

The rest of the paper is summarized as follows. Section 2 of this paper, discusses about the perviously developed approaches or frameworks used for malware detection along with the gaps present in the literature. Also, this section provides a brief structure of our proposed model based on the gaps in the literature. Section 3, explains the different feature raking approaches. Section 4, explains the different feature subset selection approaches used in this paper. Section 5 explains the LSSVM having different kernels to detect malware. In Section 6, we discuss different comparison methods of the proposed framework with existing techniques available in the literature. Performance parameters used for evaluation in this study are mentioned in Section 7. Sections 8 and 9 give the experimental setup of our proposed framework and the outcomes. In Section 10, we have discussed threats to validity and summarized our work with future scope in Section 11.

2 Related work and overview of proposed framework

Approaches or frameworks which were developed by the previous researchers to detect malware from Android apps are presented in this section. To find and overcome the gaps in the existing approaches, we divide this section in to two subsections. In the first subsection, we discuss about the frameworks or approaches, developed in the literature. In the second subsection, we first discuss about the data sets used in earlier studies and then we present the description about the collection of Android apps, formulation of our data set, extraction and formulation of feature sets, capability of features, feature selection approaches implemented in the literature. The research questions answered in this study are also formulated in this section.

2.1 Related work

In this subsection, we discuss about the analysis and its types which are used for Android malware. Later, we discuss about the detection techniques which were suggested by the previous researchers and academicians.

2.1.1 Analysis of Android apps

There are three different ways to carry out the analysis of Android apps i.e., static [29, 50, 72], dynamic [29] and hybrid [9, 34]. Static analysis is the one which analyzes the

app without executing it. In dynamic analysis, it analyzes the app during its execution. The hybrid approach is the combination of both the static and dynamic analysis. Petsas et al. [68] have explored that malicious apps targeting the Android platform can evade dynamic analysis. They applied and tested heuristics of sophistication by integrating them in existing malware samples, which attempt to conceal the presence when examined in an emulated environment. Bläsing et al. [12] suggested Android Application Sandbox (AASandbox) that execute on both static and dynamic analysis to automatically identify the suspicious apps from Android. In this study, we perform dynamic analysis of Android apps to build a malware detection model.

2.1.2 Android Malware detection

Chen et al. [18] suggested Pegasus which use the app permissions to detect malware apps. They formed Permission Event Graph (PEG) by using the fundamentals of static analysis and applied models of the APIs. Peiravian and Zhu [67] employed machine learning methods to detect malicious Android apps. They perform experiment on 1200 real malware-infected apps and validated their performance. Chakradeo et al. [17] introduced Mobile Application Security Triage (MAST), a framework which supports to manage malware resources toward the apps by the most significant potential to exhibit malicious behavior. The MAST is a statistical method that measures the correlation between multiple categorical data using Multiple Correlation Analysis (MCA).

Wang et al. [84] studied the risk of single permission and the group of permissions. They employ feature ranking methods to rank individual Android permissions based on the risk involved. Enck et al. [25] build a framework named Kirin which used the principles of light-weight certification of apps to detect malware at the time of installation. Ongtang et al. [65] presented Secure Application INteraction (Saint), that governs installation and run-time permissions. Grace et al. [35] proposed Woodpecker which examined the Android permission-based security model applied to pre-installed apps. Bugiel et al. [13] developed a model for a policy-driven and system-centric runtime monitoring of communication channels among apps at multiple layers. Zhou et al. [98] presented a systematic characterization of existing Android malware, such as, collecting charges from the devices by subscribing the services and mistreating SMS related Android permissions. Barrera et al. [8] proposed permission-based security models which helps to control access to different resources of system. They presented a methodology by doing empirical analysis of 1,100 Android apps for permission-based security models which make unique usage of self-organizing maps.

In recent study [52] Papilio introduced, a new approach for visualizing permissions of real-world Android apps. They build a new specific layout approach that includes node-link diagrams, matrix layouts and aspects of set membership. Matsudo et al. [61] presents a system model for supporting users' approval decision when an app is installed. They introduced a reputation-based security evaluation first, which analyzes permissions to judge app is malicious or not. Arp et al. [4] proposed DREBIN, a lightweight approach for the detection of Android malware. They combined concept of machine learning and static analysis, which makes malware development better. DERBIN can scan a number of apps and can protect users to install apps from untrusted sources. Jeon et al. [40] address the issues of finer-grained permissions of Android. There proposed framework was based on four major groups of Android permissions and experiments were performed by taking top Android apps to differentiate between benign or malware-infected apps. PUMA presented in [74], is a new framework for detecting malware-infected Android apps by implementing machine learning algorithms after analyzing the extracted permissions from the Android apps itself.

Grace et al. [34] developed RiskRanker, a proactive approach to accurately and scalably sift over a number of apps in existing App stores, to spot zero-delay malware. They conclude that 118,318 apps among 322 zero-day specimens from 11 distinct families were successfully discovered. TaintDroid [26] is a information flow tracking tool that can concurrently track multiple sources of sensitive data. A new model to protect smartphones was discussed in [70]. This model execute attack detection on a remote server where the implementation of the app on the smartphone mirror in a virtual machine. Schmidt et al. [5] presented anomaly detection using machine learning to monitor system-based information and system gathering behavior that is processed by a remote system. Zheng et al. [95] focused on the demanding task of triggering a particular behavior through automated UI interactions. They proposed a hybrid analysis approach to display UI-based trigger conditions in Android apps. To discover malware at kernel-level and user-level, a technique, named MADAM, has been developed in [24], which is capable to distinguish malware or benign apps. A fine-grained dynamic binary instrumentation tool named as DroidScope is presented in [91], for Android that reconstructs two levels of semantic information i.e., Java and operating system. A framework to monitor system calls named as Crowdroid is introduced in [14]. Crowdroid can see the track of information flows and API analysis which paid great impact to find malware activities in the network.

A root privilege management scheme called Root Privilege Manager (RPM) were proposed by [80]. It prevents Android apps from the risk raised by the permissions i.e., normal or dangerous. Wang et al. [82] analyses the used permissions and support-based permissions candidate method to detect Android malware. A hybrid feature selection approach which work on Rough Set Quick Reduct algorithm to detect malware was proposed in [10]. Wang et al. [85] collected 11 kind of static features by extraction from each type of app to characterize its behavior. By collecting the behavior, they applied classification algorithms to categorize malware and benign apps. Kirubavathi et al. [46] proposed a structural-based analysis learning approach, which accepts machine learning algorithms to detect malware and benign apps. They adopt botnet linked patterns of requested permissions as a feature to evaluate benign and malware apps. Jerlin et al. [41] suggested a new approach to detect malware by using its Application Programmable Interfaces (APIs). They adopt upper and lower boundaries as one of its feature to detect malware from Android. Mahindru and Singh [60] applied supervised machine learning algorithm on 172-permissions extracted during its installation and start-up time from Android apps.

Xiao et al. [88] proposed an approach that was based on deep learning to distinguish between benign and malware apps. In their approach, they consider system call as feature and trained it with the help of Long Short-Term Memory (LSTM) classifier. In their study, they trained LSTM models with system call sequences from malware and benign apps. Experiments were performed on 3567 malware-infected and 3536 benign apps and achieved recall of 96.6%. Mahindru and Sangal [54] proposed a framework DeepDroid that works on the principle of deep learning. They extract permissions and API calls as features from collected Android application packages (.apk). To select significant features to develop malware detection model six distinct feature ranking approaches are applied on extracted features. Experiments were performed on 1,00,000 benign apps and 20,000 malware-infected apps. Framework developed using Principal component analysis (PCA) as feature ranking approach achieved a detection rate of 94%. Letteri et al. [49] proposed a botnet detection methodology for *internet of things* (IOT) based on deep learning techniques, tested on a new, SDN-specific data set with a high (up to 97%) classification accuracy.

Devpriya and Lingamgunta [23] proposed a novel hash-based multifactor secure mutual authentication scheme that includes hashing properties, certificates, nonce values, traditional user ids, and password mechanisms that resist MITM attacks, replay attacks, and forgery attacks.

Ma et al. [53], presented Android malware detection model based on the principle of API information. In their study, with the help of API information they construct three distinct data sets that are related to boolean, frequency and time-series. Based on these three data sets, three distinct detection models are developed. Experiments were performed by using 10010 benign and 10683 malware apps and achieved an accuracy of 98.98% by considering an ensemble approach. Mahindru and Sangal [59] proposed PerbDroid that developed by using features selected by feature ranking approaches and deep learning as machine classifier. Experiments were performed on 2,00,000 distinct Android apps and achieved a detection rate of 97.8%. Wang et al. [86] proposed a hybrid model based on convolutional neural network (CNN) and deep autoencoder (DAE). To improve the accuracy of malware detection model, they employed multiple CNN to select features from high-dimensional features of Android apps. Experiments were performed on 10,000 benign and 13,000 malware-infected apps and trained it with the help of serial convolutional neural network architecture (CNN-S). Mahindru and Sangal [56] proposed malware detection model with semi-supervised machine learning techniques. They applied LLGC algorithm on 2,00,000 distinct Android apps and achieved an accuracy of 97.8%.

Yamaguchi and Gupta [90] discussed properties of IOT device which make it more vulnerable for malware attacks i.e., large volume and pervasiveness. In their study, they proposed method to mitigate the attack on IOT based devices. Gupta et al. [36] proposed a book that is related to security measure and challenges faced by different communication devices. They also discussed different methods to mitigate the attacks. In [32], it was seen that feature selection approach paid a great effect in developing the model. Authors implemented Principal Component Analysis (PCA) to reduce the complexity of the model. With the advancement in the machine learning algorithms like SVM [20, 94], Deep learning model [32, 38, 96], it not only helped in detecting intrusion detection [45], cyber attacks but it also helped in health sector and in wireless routing too. Distinct researchers applied deep learning model and hybrid methods [32, 38, 96] in their study and achieved remarkable results.

Arora et al. [3] proposed PermPair, in which they construct and compare the graphs by extracting permissions from benign and malware-infected apps. Empirical result reveals that proposed malware detection model achieved an accuracy of 95.44% when compared to other similar approaches and favorite mobile anti-malware apps. Mahindru and Sangal [55] proposed DLDroid malware detection model, that is based on feature selection approaches and Deep Neural Network (DNN) machine learning algorithm. In their study, they collected Android apps that are developed during COVID-19. Experiments were performed on 11,000 distinct Android apps and model developed using DNN and Rough set analysis achieved a detection rate of 97.9% when compared to distinct anti-virus scanners available in the market.

Table 1 describes the brief details of some existing Android malware detection techniques present in literature. It also includes the type of monitoring and type of analysis used for these techniques. The conclusions made from these techniques are presented in the last column of the table.

Table 1 Brief description of some existing Android malware detection frameworks or approaches

Framework / Approaches	Detection type	Monitoring type	Analysis type	Observations
AASandbox (2010) [12]	Dynamic	System and Library Calls	Clustering	Clustering is used to trained the data, but testing is not performed
TaintDroid (2010) [26]	Dynamic	Program Traces	Expert	Tracking of label-based variables, methods, files and IPC by dynamic analysis
Paranoid Android (2010) [70]	Dynamic	Program Traces	Expert	Detection on a remote server where the execution of software on phone as a virtual machine
Schmidt et al. (2011) [5]	Static and Dynamic	System Calls	Clustering	SVM-light machine learning technique applied
Crowdroid (2011) [14]	Dynamic	System Calls	Clustering	Clustering based K-means applied on experimental and Wild malware
Woodpecker (2012) [35]	Static	Permissions	Dependency Graphs	Uses CFG for detecting explicit leakages and permission analysis for implicit capability leakage
RiskRanker (2012) [34]	Static	Instructions, Permissions and API calls	Dependency Graphs	A set of pre-defined malicious operations to rate apps in which family they belong
SmartDroid (2012) [95]	Static and Dynamic	Program Traces	Dependency Graphs	Improved detection by generating UI-based trigger condition
MADAM (2012) [24]	Dynamic	Kernel-level and User-Level	Machine Learning	5% FP rate and 93% detection rate
DroidScope (2012) [91]	Dynamic	Kernel-level and User-level	Unavailable	Evaluation showed the benefit of dynamically disabling JIT for targeted analysis
AppGuard (2012) [7]	Dynamic	Program Traces	Unavailable	Analysis is done offline, prior to repackaging the app
Andromaly (2012) [76]	Dynamic	Behavioural Monitoring	Machine Learning	Training Method: Classification with labeled data

Table 1 (continued)

Framework / Approaches	Detection type	Monitoring type	Analysis type	Observations
Aurastium (2012) [89]	Dynamic	Behavioural	Repackaging	100% success rate
TstructDroid (2013) [77]	Dynamic	Process Control Block	Machine Learning	Machine learning classification technique is applied
AppsPlayground (2013) [71]	Dynamic	System Calls and Program Traces	Unavailable	Heuristic based UI interaction approach is followed
AppProfiler (2013) [73]	Static and Dynamic	Program Traces and API Calls	Expert	Using signature API calls are analysed
Andrubis (2014) [51]	Static and Dynamic	Dalvik and System level	Expert	Used data set of 1,000,000 Android apps where 40% are malicious apps
Androguard (2015) [22]	Static	Disassemble and Decompile apps	Control Flow Graphs	Open-source, finds differences and similarities of two suspected clones
CopperDroid (2015) [79]	Dynamic	System Call	Hierarchical	2,900 real-world malwares-infected apps used
MADAM (2016) [75]	Static and Dynamic	Multiple Features	Machine Learning	11,000 Android apps were considered
HinDroid (2017) [39]	Dynamic	API call	Machine Learning	Limited data set used
DroidCat (2018) [15]	Dynamic	Interprocess Communication	Machine Learning	34,343 apps were used
MalDozer (2018) [44]	Dynamic	API calls	Machine Learning	Limited data set were used
DroidDet (2018) [100]	Static	API calls and Permissions	Machine Learning	2,130 Android apps were considered.
DeepDroid (2019) [54]	Dynamic	API calls and Permissions	Machine Learning	1,20,000 Android apps were used
PerbDroid (2020) [59]	Dynamic	System call and permissions	Machine Learning	2,00,000 distinct Android apps utilized
GADroid (2020) [57]	Dynamic	System Call and permissions	Machine Learning	Consider 25,000 malware-infected apps

2.2 Gaps and overview of our proposed framework

In this subsection of the paper, we discuss about the gaps that are present in the previous studies and how we can overcome these gaps while developing an efficient and effective Android malware detection framework.

2.2.1 Gaps present in the previous frameworks/approaches

The research done earlier in this field had the following limitations: use of limited data set, high computation burden and unable to detect sophisticated malware. To overcome the first limitation, in this study, we collect 2,00,000 Android apps which belong to thirty different categories from different promised repositories mentioned in Table 3. Further, to select significant features which help to reduce computation burden, we implement ten distinct feature selection approaches on extracted feature data set (i.e., Permissions, API calls, number of user download the app and rating of an app). Next, selected features are considered as input to build model by using distinct machine learning algorithms so that suitable framework is build to detect malware from real-world apps. In the previous studies, developed frameworks/approaches were not tested on real-world apps.

2.2.2 Description of the collected Android apps

Pervious frameworks or studies mentioned in Table 2, used only limited data sets of Android apps to examine its associations with malware or benign class. Therefore, it is not able to draw generic conclusion relevant to all Android apps and its system. To overcome this gap, we collect apps that belongs to thirty different categories which are used to generalize and strengthen our outcomes. In this study to develop efficient and effective Android malware detection model, we collect Android Application packages (.apk) from different promised repositories. We collected 2,00,000 of .apk files, from Google's

Table 2 Previous developed studies/framework for Android malware detection

Previous Work	Data set used	Implementation
Kirin [25]	311 apps	On-device
Scandroid [31]	—	On-device
TaintDroid [26]	1,100 apps	Off-device
AASandbox [12]	150 apps	Off-device
Crowdroid [14]	Self-written malware apps	Off-device
DroidMOSS [97]	68,187 apps	Off-device
Andromaly [76]	Self-written malware apps	On and Off-device
AndroSimilar [27]	7324 apps	Off-device
PUMA [74]	1811	Off-device
CopperDroid [79]	2900 + apps	Off-device

Table 3 Categories of .apk files belong to their respective families (.apk)

ID	Category	Normal	Trojan	Backdoor	Worms	Botnet	Spyware
D1	Arcade and Action (AA)	6291	440	100	204	130	600
D2	Books and Reference (BR)	5235	200	250	56	150	150
D3	Brain and Puzzle(BP)	4928	820	54	28	50	50
D4	Business (BU)	8308	152	150	150	22	22
D5	Cards and Casino(CC)	2886	76	65	81	100	44
D6	Casual(CA)	2010	321	69	46	150	140
D7	Comics (CO)	7667	65	95	35	3	0
D8	Communication (COM)	8414	250	50	500	3	3
D9	Education (ED)	8744	560	68	50	50	68
D10	Entertainment(EN)	4222	500	500	500	100	42
D11	Finance (FI)	3999	50	200	99	65	92
D12	Health and Fitness(HF)	8551	98	65	45	140	140
D13	Libraries and Demo (LD)	5655	70	100	100	6	500
D14	Lifestyle (LS)	7650	155	200	100	193	192
D15	Media and Video (MV)	8019	100	123	162	450	71
D16	Medical(ME)	6000	12	13	12	24	25
D17	Music and Audio (MA)	8621	65	100	65	165	165
D18	News and Magazines (NM)	8164	100	100	100	100	32
D19	Personalization (PE)	4334	500	42	500	200	22
D20	Photography (PH)	9133	100	120	50	96	500
D21	Productivity (PR)	9850	100	516	250	250	62
D22	Racing (RA)	7766	50	100	210	100	180
D23	Shopping (SH)	2673	100	100	120	150	50
D24	Social (SO)	6159	100	50	210	150	150
D25	Sports (SP)	2669	100	240	100	450	112
D26	Sports Games (SG)	3889	100	145	145	650	198
D27	Tools (TO)	3346	120	500	550	475	563
D28	Transportation (TR)	3796	2	500	100	100	20
D29	Travel and Local (TL)	3180	500	220	150	48	100
D30	Weather (WR)	2841	120	23	700	50	25

Malware families are identified by anti-virus scanners

play store, hiapk,¹⁴ appchina,¹⁵ Android,¹⁶ mumayi,¹⁷ gfan,¹⁸ slideme¹⁹ and pandaapp.²⁰ Among these 2,00,000 benign .apk files, 1,75,000 are distinct. Further, the features are

¹⁴<http://apk.hiapk.com/>

¹⁵<http://www.appchina.com/>

¹⁶<http://android.d.cn/>

¹⁷<http://www.mumayi.com/>

¹⁸<http://apk.gfan.com/>

¹⁹<http://slideme.org/>

²⁰[http://download.pandaapp.com/?app\\$=\\$soft&controller\\$=\\$android#.V-p3f4h97IU](http://download.pandaapp.com/?app$=$soft&controller$=$android#.V-p3f4h97IU)

extracted after deleting viruses infected apps, reported by VirusTotal and Microsoft Windows Defender. VirusTotal identify malware affected apps by antivirus engines, it contains the definition of 70 antivirus softwares. A total of 35,000 malware samples, are collected from three different promised repositories. Kadir et al. [43], introduced Android sample set of 1929 botnets, consisting of 14 distinct botnet families. Android Malware Genome project [98] contains a data set of 1200 malware samples that cover the currently present Android malware families. We collected about 17,871 samples from AndroMalShare²¹ along with their package names. After removing duplicate packages from the collected data set, we have 25,000 unique malware samples left in our study. Both benign and malware apps being collected from the above mentioned sources at the end of December 2018. Table 3 shows the number of .apk files belonging to different categories i.e., business, comics, communication, education and so on. To better differentiate between benign and malware apps we consider .apk files belonging to normal, trojan, backdoor, worms, botnet and spyware families²² mentioned in Table 3.

2.2.3 Formulation of data set

After collecting a unique samples of .apk files from various sources mentioned in previous subsection, we extract permissions and API calls from each of the .apk file. Extraction of permissions and API calls have been performed with the help of an emulator (in our study we use Android studio). Emulator provides the same API level and execution environment as our smartphones provide to us. In our study, to extract permissions and API calls from Android apps we use Android system version 6.0 Marshmallow (i.e., API level 23) and form our data set for experiments. Previous developed frameworks or approaches used the previous version of Android to extract features from them. There are two reasons for selecting this Android version: first, it asks the user to revoke or grant the permission to use the resources of smartphones and second it covers 28.1% of Android devices which is higher than other versions present in the market²³. A flowchart showing that how an app gets installed and demand permissions on Android 6.0 is presented in Fig. 2. When we start the installed app at the very first time, it demands some of the permissions which are required by the app to function properly. Next, SDK level is checked if the version of SDK is ≥ 23 then Android ask from the user to grant or revoke the permission to the app. If the user grant the permissions to the app, then call command is executed otherwise not. This facility was not available in the earlier versions of Android.

Figure 3 demonstrates the phases which are followed in extracting features from Android apps. In the first phase, to extract features from collected .apk, we perform dynamic analysis by using Android studio as an emulator. To extract permissions and API calls, we use Android 6.0, i.e., Marshmallow mentioned above. Further, we write a program in java language and extract permissions and API calls from them and save into the .csv file [58] that are publicly available for researchers and academicians.²⁴ These permissions are demanded by apps during their installation and start-up time. By using the same process again and again, we extract permissions from 2,00,000 different Android apps and record them in the .csv file format. Previous researchers used limited set of features to develop a model for

²¹<http://202.117.54.231:8080/>

²²Malware families are identified by VirusTotal.

²³<https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/>

²⁴<https://data.mendeley.com/datasets/9b45k4hkdf/1>

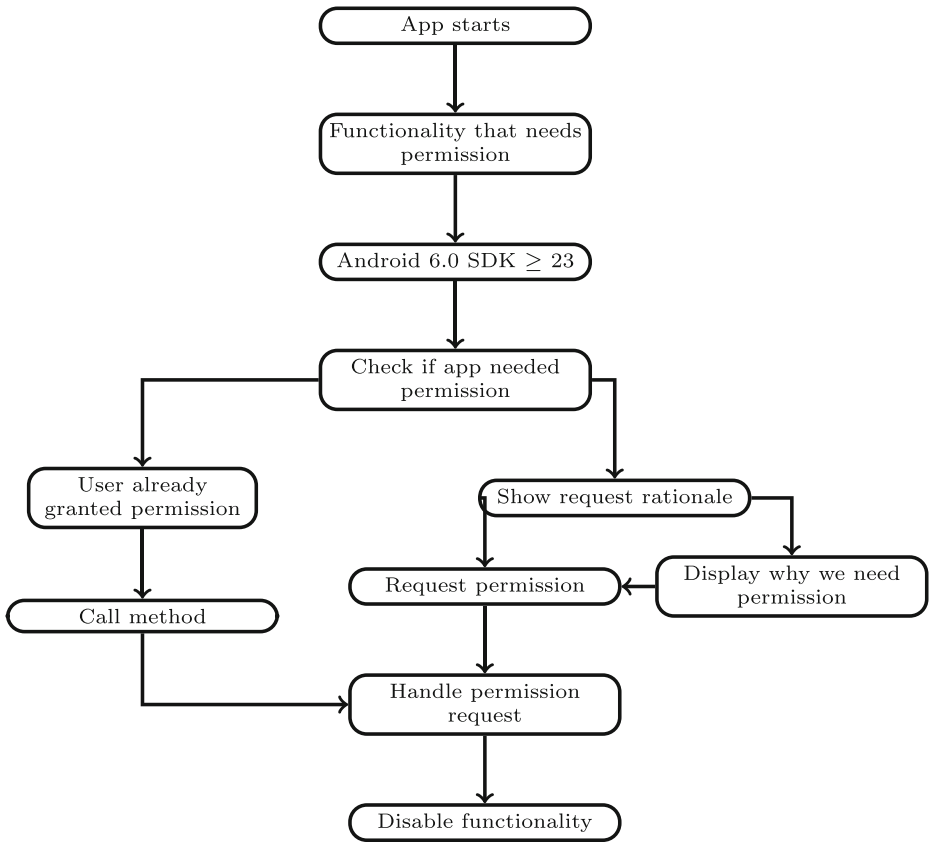


Fig. 2 Android API level 23 working flow chart

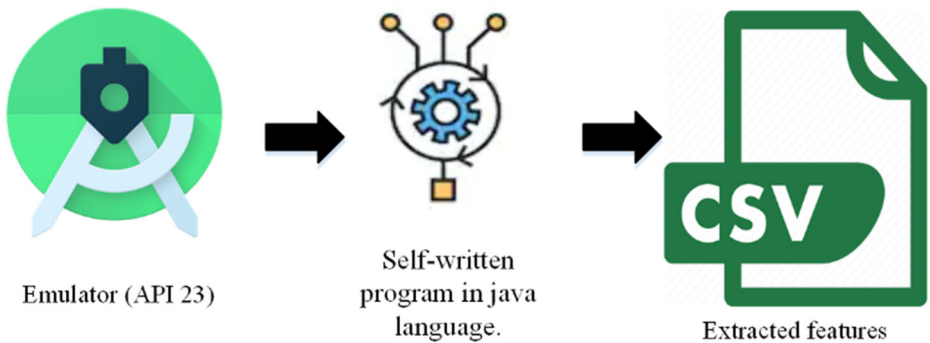


Fig. 3 Extraction of features from .apk files

malware detection. To overcome this gap, in this study we collect 1532 permissions, 310 API calls, number of user download the app and rating of an app²⁵ which helps to build a model by using feature selection approaches. Hence, each of the collected app can be represented as a 1844-dimensional Boolean vector, where “1” implies that an app requires the permission and “0” implies that the permission is not required. It is very common that distinct apps may request the similar set of permissions for its execution. Permissions overview given by Google²⁶ is used to describe the behavior of a permission i.e., “dangerous” or “normal”.

2.2.4 Formulation of feature sets

Several approaches had been developed for Android malware detection [1, 8, 87]. In this study, we divide the extracted API calls and permissions in to thirty different feature sets which helps

in developing malware detection model.²⁷ Table 4 displays the basic description of the feature sets that are considered in our work.

2.2.5 Capability of features

The objective of this work is to study, the relationship among the features and its associated classes (i.e., benign or malware) which helps in malware detection. In this paper, malware is considered as dependent variable and feature sets are taken into consideration as independent variables. Malware is supposed to be a work of numerous set of features for instances S1, S2, S3, S4, S5, ..., up to S30. To investigate the efficacy of the feature sets utilized, they are further classified into three different groups which are written as under:

- a. **Analysis 1 (A1):** The efficacy of feature sets is utilized to dat the level of class. The relationship among malware and feature sets is possibly outlined as below:

$$\text{Malware} = f(S1, S2, S3, S4, S5, \dots, S30)$$

- b. **Analysis 2 (A2):** Reduced feature attributes attained by utilizing *feature ranking* approaches are considered as an input for building a malware detection model for Android. To rank the features based on their performance, in the present work we consider six distinct feature ranking approaches. According to the study, [33, 76], it may be relevant to choose top $\lceil \log_2 q \rceil$ set of features by utilizing feature ranking approaches. In the present study, we also use top $\lceil \log_2 q \rceil$ set of features to detect either the app is benign or normal. Here “ q ” indicates the set of features present in a novel data set. The association can be presented in the following way:

$$\text{Malware} = f(\text{reduced set of features utilizing feature ranking approaches})$$

²⁵ In this study, we use the Min-max normalization approach to normalize the data. This approach is based on the principle of linear transformation, which bring each data point D_{q_i} of feature Q to a normalized value D_{q_i} , that lie in between 0 – 1. Following equation is considered to find the normalized value of D_{q_i} :

$$\text{Normalized}(D_{q_i}) = \frac{D_{q_i} - \min(Q)}{\max(Q) - \min(Q)},$$

where $\min(Q)$ & $\max(Q)$ are the minimum and maximum significance of attribute Q , respectively.

²⁶ <https://developer.android.com/guide/topics/permissions/overview>

²⁷ Name of the extracted feature sets are available at url: <https://github.com/ArvindMahindru66/Computer-and-security-dataset> for reserachers and academicians.

Table 4 Formulation of Sets containing (permissions, API calls, Number of user download the app and rating of the app) as an features

Set number	Description	Set number	Description
S1	Related to SYNCHRONIZATION_DATA	S2	Related to CONTACT_INFORMATION
S3	Related to PHONE_STATE and PHONE_CONNECTION	S4	Related to AUDIO and VIDEO
S5	Related to SYSTEM_SETTINGS	S6	Related to BROWSER_INFORMATION
S7	Related to BUNDLE	S8	Related to LOG_FILE
S9	Related to LOCATION_INFORMATION	S10	Related to WIDGET
S11	Related to CALENDAR_INFORMATION	S12	Related to ACCOUNT_SETTINGS
S13	Related to DATABASE_INFORMATION	S14	Related to IMAGE
S15	Related to UNIQUE_IDENTIFIER	S16	Related to FILE_INFORMATION
S17	Related to SMS_MMS	S18	Related to READ
S19	Related to ACCESS_ACTION	S20	Related to READ_AND_WRITE
S21	Related to YOUR_ACCOUNTS	S22	Related to STORAGE_FILE
S23	Related to SERVICES.THAT_COST_YOU_MONEY	S24	Related to PHONE_CALLS
S25	Related to SYSTEM_TOOLS	S26	Related to NETWORK_INFORMATION and BLUETOOTH_INFORMATION
S27	Related to HARDWARE_CONTROLS	S28	Related to Default group
S29	Contain info. Related to API calls	S30	Contain info. Related to rating and downloads

- c. **Analysis 3 (A3):** Feature subset selection approaches are considered to reduce the size of feature sets which further can be considered as an input to build a model for detecting malware from Android. In the present research, we consider four distinct feature subset selection approaches to discover abbreviated subset of features. The association can be presented in the following way:

$$\text{Malware} = f(\text{reduced set of features utilizing feature subset selection approaches})$$

2.2.6 Feature selection approaches

On the basis of Table 5, it is seen that in previous studies a number of authors applied different feature ranking approaches to detect malware from Android apps and achieved a good detection rate. This indicates that the outcome of malware detection model rely on the features that are taken as an input to design a model. Selecting the suitable feature sets is essential for data preprocessing task in machine learning. In the field of malware detection, some researchers have used selection approaches to select appropriate set of features. Allix et al. [2] performed an empirical validation of Android apps for malware detection. They performed the static analysis of Android apps' bytecode to withdraw an illustration of the program control-flow graph (CFG). Further, the extracted CFG is stated as character strings in their study on establishing similarity between Android app. They derived that string representation of the CFG is an abstraction of the app's code that keeps information related to structure of the code, on the other hand they discards low-level details such as variable names or registered numbers. This is a desirable property in the context of malware detection. They performed malware detection on 50,000 Android apps. Shabtai et al. [76] investigated whether the feature extraction at app level, operating system level, scheduling level etc. helps them to detect malware or not. They performed experiment on a small number of Android apps, used less number of features and implemented less number of feature selection approaches to detect malware. Also, none of the researches done earlier in this direction have used feature subset selection approaches. So, in this paper we implemented ten distinct types of feature selection approaches on a large collection of 1842 features (divided in to thirty distinct feature sets) to identify the best subset of features which assist us to detect malware detection with better detection rate and also minimize the figure of misclassification errors. Feature ranking approaches and Feature subset selection approaches can be defined in the following manner:

- *Feature ranking approaches:* These approaches, use certain conclusive elements to rank the features. Further, on the basis of their ranks appropriate features can be selected to build the model.
- *Feature subset selection approaches:* These approaches aim to search subset of features which can have good detective capability.

2.2.7 Research questions

To develop a malware detection model for Android malware detection from real-world apps with better detection rate and to cover the gaps in the literature (i.e., selection of right feature sets to develop a model, implementation on large collection of data set and implementation of proposed framework on real-world apps), we consider the following research questions in this research paper:

Table 5 Feature selection method, technique and features used in the literature for malware detection

Author/Approach	Feature selection method used	Techniques used	Features used
Information gain			
	Naive Bayes (NB),	System call K-nearest Neighbour (KNN), Decision Tree (J48), Multi-Layer Perceptron (MLP) and Random Forest (RF)	
Yerima et al. [92]	Mutual Information	Bayesian classification	Permissions and API calls
Yerima et al. [93]	Information gain	Bayesian classifier	Permissions and API calls
MKLDroid [62]	Chi-squared	kernel methods	Permissions, API calls, etc.
Allix et al. [2]	Information gain	RandomForest, C4.5, RIPPER and Support Vector Machine (SVM)	textual representations of basic blocks extracted from the Control-Flow Graph of apps' bytecode
Andromaly [76]	Chi-square, Fisher score and Information gain	Decision Tree (J48), Naive Bayes (NB), Bayesian network, k-Means, Histogram or Logistic Regression	Feature extraction at app level, operating system level, scheduling level etc.
Azmoodeh et al. [6]	Information Gain	Deep Eigenspace learning approach	Operational Code (OpCode) sequence Syntax features
Chen et al. [19]	Manual pruning along with information gain	Support Vector Machine (SVM), Random Forest(RF), and K-Nearest Neighbor (KNN)	
Narudin et al. [63]	ClassifierSubsetEval	Bayes network, Multi-layer perceptron, J48, K-Nearest neighbours and Random forest	Network traffic features
ANASTASIA [30]	Ensemble of randomized decision trees (i.e., Extra Trees-Classifer)	Support Vector Machine (SVM), Decision Tree (J48), Logistic Regression, Naive Bayes (NB), Random Forest(RF), K-Nearest neighbours, AdaBoost and Deep Learning	System command Permissions and API calls

RQ1. Which malware detection model is most appropriate to detect malware from real-world apps?

This question helps in finding the most appropriate model which is suitable for malware detection in Android. In this work, we build 30 distinct models by considering ten distinct feature selection approaches and three different machine learning techniques. Further, to identify that which model is more appropriate for malware detection we consider two performance parameters i.e., F-measure and Accuracy in our study.

RQ2. Whether the presented malware detection framework is effective or not to detect malware from Android devices?

The goal of this question is to investigate the performance of our malware detection approach. For this, we compare the performance of our developed model with some existing techniques available in the literature.

RQ3. Does a subset of feature perform better than all extracted features for the task of detecting the app is malware or not?

The aim of this question, is to evaluate the features and investigate their relationship among benign and malware apps. Distinct kinds of feature reduction approaches are being considered for finding subset of features which are able to detect either the app is benign or not.

RQ4. Among different implemented feature ranking approaches which approach work best for the task to detect either the Android app belong to benign or malware class?

In feature ranking approach, efficiency of the machine learning algorithms is affected by the characteristics and nature of the malware data set. Distinct approaches are being implemented with various criteria to rank the collected feature sets. Two distinct performance criteria i.e., F-measure and Accuracy are considered in this study, to compare distinct feature-ranking approaches.

RQ5. Among applied feature subset selection approaches which approach performs foremost for the task of detecting malware from Android apps?

To determine the subset of features which are appropriate to detect either the Android app is benign or malware we consider feature subset selection approaches. In this work, we compare distinct approaches by using two performance criteria i.e., F-measure and Accuracy.

RQ6. How do the feature subset selection approaches compare with feature ranking approaches?

In this paper, pair-wise *t*-test being used to determine either the feature subset selection approaches are appropriate than feature ranking approaches or both of them behave equally well.

RQ7. Do the feature selection approaches effect on the outcome of the supervised machine learning approaches?

It is seen that number of feature selection approaches perform extremely well with specific supervised machine learning methods. Therefore, in this research work distinct feature selection approaches are evaluated using distinct supervised machine learning approaches to measure their performance. Further, it also emphasizes on variation of performance of supervised machine learning approach over distinct supervised machine learning approaches.

3 Feature ranking approaches

These approaches rank features separately without applying any training algorithm. Ranking of features depends upon their score. On the basis of our investigation of the previous studies, the majority of approaches are capable to calculate the grading of every feature. In this research, we employ six different ranking approaches to rank the features. Feature ranking approaches are explained below:

3.1 Gain-ratio feature selection

In this selection approach, feature ranking work on the prediction of the gain-ratio in relation to the class [64]. The “Z” known as the gain-ratio of feature is determined as:-

$$\text{Gain-Ratio} = \frac{\text{Gain}(Z)}{\text{SplitInfo}_Z(X)}, \quad (1)$$

where $\text{Gain}(Z) = I(X) - E(Z)$ and X depicts the set including m numbers of instances with n different classes. The forthcoming statistics necessary to categorize a given sample is calculated by utilizing succeeding equation:

$$I(X) = - \sum_{i=1}^m P_i \log_2(p_i). \quad (2)$$

Here in this equation P_i is the chance that a random sample can be a member of class C_i and is measured by z_i/z .

The number of instances is given by z_{ij} of class C_i in subset N_j . The foreseen knowledge is relying on the partition of subsets by F , and is presented by

$$E(Z) = - \sum_{i=1}^M I(X) \frac{n_{1i} + n_{2i} + \dots + n_{mi}}{n}. \quad (3)$$

$\text{SplitInfo}_F(X)$ is measured by utilizing following equation:

$$\text{SplitInfo}_F(X) = - \sum_{i=1}^t \frac{|X_i|}{X} \log_2\left(\frac{|X_i|}{X}\right) \quad (4)$$

The value of $\text{SplitInfo}_F(X)$ show us the details achieved by dividing the data set of training X into t portions equivalent to t results of a test on the attribute Z .

3.2 Chi-Squared test

This test is employed to examine the self-determination among two events [69], and in our work, ranking of features is predicted by the significance of its statistic in relation to the class. Higher the calculated value implies the denial of the outliers and consequently these features can be analyzed as better relevance to detect malware from Android apps.

3.3 Information-gain feature selection

In Info-gain features are selected on its relation with respect to the class [64].

3.4 OneR feature selection

OneR feature selection approach is used for grading the features [64]. To rank individual features it utilizes the classification mechanism. In it valuable features are considered as constant ones and divide the set of values into a few dissociate intervals made by straightforward method. In this study, we consider features with better classification rates.

3.5 Principal Component Analysis (PCA)

Reduction of attribute is accomplished by implementing PCA on our collected data set. PCA helps in transforming a high dimension data space into a low dimension data space. Features which are present in low dimension have extreme importance in detecting malware [83]. Correlation among several features are high, so PCA is utilized to relocate these features that are not extremely correlated. The features obtained are named as principal component domain features. Further, to identify significant patterns in the data a small value of principal components is sufficient. The detailed phases of PCA are demonstrated in Fig. 4.

Feature data set is collected in the form of $m * n$ matrix, that contains n number of data sample and m number of extracted features. In the second phase, normalization of the feature data set is performed by using equation

$$\mu_j = \frac{1}{n} \sum_{i=1}^n x_i^j$$

and replace x^j with $(x^j - \mu_j)$. Next, we calculate eigen value and eigen vector by using matlab environment. Next, to select first k number of principal components from the covariance matrix we performed following steps

```

while(i=1 to m) do evaluate cumvar =  $\sum_{i=1}^k \lambda_{ii}$ 
 $\sum_{i=1}^m \lambda_{ii}$ 
if (cumvar  $\geq$  0.99) or (1 - cumvar  $\leq$  0.01)
return k 99% of variance is retained
end if
end while

```

cumvar denotes (cumulative variance) and (λ) represents eigen values sorted in descending order.

After evaluating this, reduced feature sets are selected for training purpose.

3.6 Logistic regression analysis

For feature ranking, Univariate Logistic Regression (ULR) analysis being considered to verify the degree of importance for every feature sets [21]. In the current work, we consider two benchmarks of LR model; to discover the importance of every feature and to rank each feature sets. Parameters for Logistic regression analysis are as follows:

1. *Value of regression coefficient*: The coefficient measure of features indicates the degree of correlation of every feature sets with malware.
2. *P-value*: P-value i.e., level of significance shows the correlation significance.

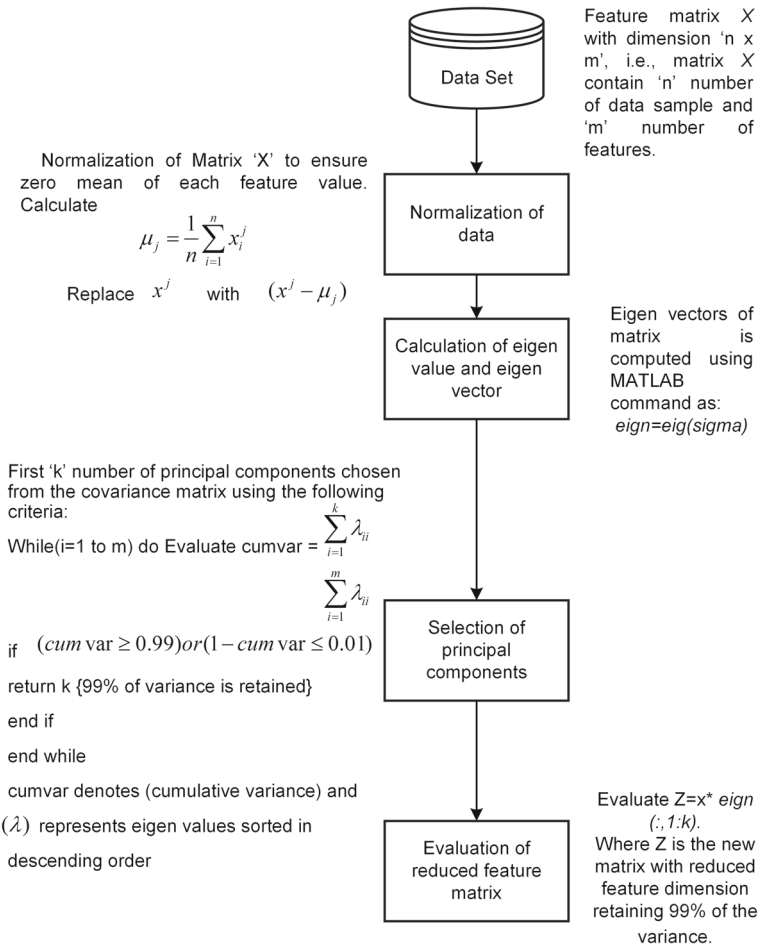


Fig. 4 Framework of PCA calculation

4 Feature subset selection approaches

These approaches are employed to detect appropriate subset of features which jointly have best detective capability. These are established on the hypothesis that developed model has better detection rate and lower value of misclassification errors when linked with few other features or when matched by itself. Several approaches are feasible to identify the right subset of features which helps in detecting malware. In this work, four distinct feature subset selection approaches are considered to calculate the score of feature. Implemented approaches are depicted below:

4.1 Correlation based feature selection

This approach is based on correlation approach which select a subset of features that are particularly related to the class (i.e., benign or malware). In this research paper, Pearson’s correlation (r : Coefficient of correlation) has been used for searching the dependency among

features. If the value of “r” is higher among the feature sets, it indicates a strong relation among these features. It further implies that, there is a statistical reason to consider those classes which are having lower (or highest) feature value with that it have lower (or highest) ranges of other highly correlated features.

4.2 Rough set analysis (RSA)

This approach is an estimation of conventional set, in terms of a joins of feature sets which provide the upper and the lower estimation of the original data set [66]. This formal estimation, depicts the upper and lower limits of the original data set. The application of this approach is in mining the data from imperfect data. This approach is used to select the reduced set of features from the extracted feature sets. RSA used three distinct notations such as approximations, reduced attributes and information system. The steps that are pursued to get reduced subset by utilizing RSA are mentioned-below and also demonstrated in Fig. 5.

- i. *Approximation*: Let $A = (C, Z)$, $X \subseteq Z$ and $Y \subseteq C$. X – topmost (XY) and X – lowermost (\underline{X}) approximations of X are utilized to estimate Y . The topmost limit includes all the objects which maybe the part to the set and the lowermost approximation includes of all objects which certainly be a part of the set. The XY and (\underline{X}) are computed by utilizing subsequent equations:

$$\bar{X}Y = \{y_i \in U \mid [y_i]_{Ind(B)} \cap Y \neq \emptyset\} \tag{5}$$

$$\underline{X}Y = \{y_i \in U \mid [y_i]_{Ind(B)} \subseteq Y\}, \tag{6}$$

where $[y_i]_{Ind(C)}$ belongs to the same class of y_i in connection $Ind(C)$.

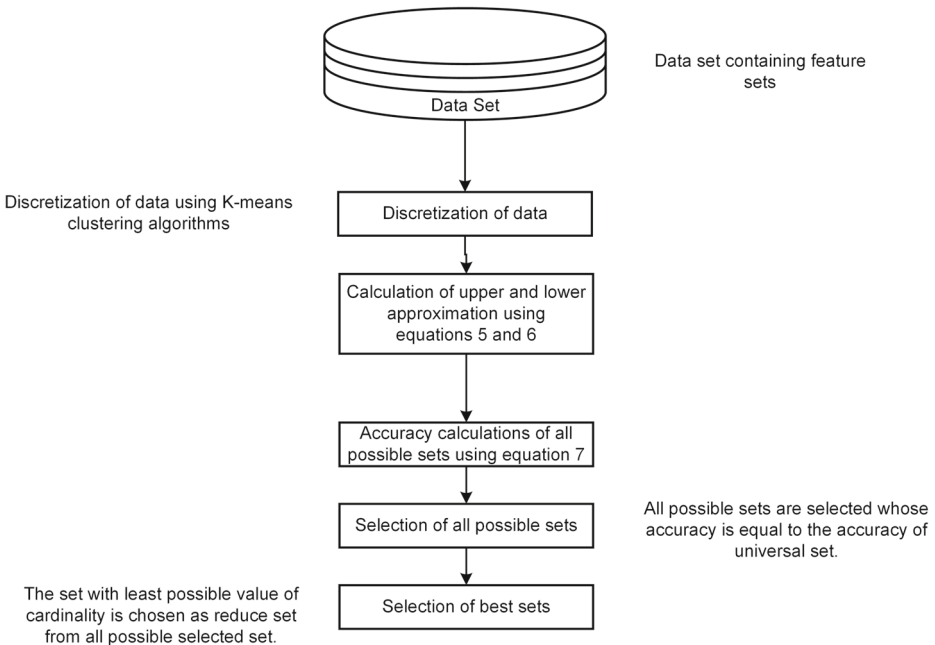


Fig. 5 Rough set theory framework

- ii. *Reduced attributes*: Correctness evaluation of the group Z ($Acc(Z)$) in $A \subseteq B$ is determined as:

$$\mu_B(A) = \frac{card(\underline{B}Z)}{card(\overline{B}Z)} \tag{7}$$

The number of features contained in the topmost or uppermost approximation of the set Z is called the cardinality of the set. Further, all possible feature sets are considered whose accuracy is equivalent to the accuracy of extracted feature sets.

- iii. *Information system*: It is determined as $Z = (C, B)$, where C is a universe including non-empty set of confined objects and B is the sets of attributes with a finite number of elements. For each $b \in B$, there exist a function $F_b : C \rightarrow V_b$, where V_b denotes the value of attribute b . For each $A \subset B$, there exists an equivalence relation known as B-indisceribility relation ($Ind(Z)$). $Ind(Z)$ can be defined as:

$$IND_A(Z) = \{(x, y) \in C^2 \mid \forall a \in Z, a(x) = a(y)\}. \tag{8}$$

4.3 Consistency subset evaluation approach

This technique provides the importance of subset of attributes by their level of consistency appearing in class values, when the training instances are applied on the subset of attributes. The consistency rate is calculated with the help of inconsistency rate, where two data elements can be considered as inconsistent if they belong to different class labels (i.e., benign or malware) but have same feature values. For this work, destination variable i.e., apps having two distinct characteristics (i.e., 0 for benign apps and 1 for malware apps). A group of feature (GF) is having Z amount of sample, there are z amount of instances in a manner that $Z = X_1 + X_2 + \dots + X_z$. Instance X_i seems in entirely A samples from which A_0 numbers of samples are marked by 0 and A_1 number of samples are marked by 1, here $A = A_0 + A_1$. If A_1 is less than A_0 , then the difference count for the instance X_i is $INC = A - A_0$. The inconsistency rate ($INCR$) of feature set is computed by utilizing succeeding equation:

$$INCR = \frac{\sum_{i=1}^z INC_i}{Z} \tag{9}$$

4.4 Filtered subset evaluation

Filtered subset evaluation is based on the principle to select random subset evaluator from data set that was gained by applying arbitrary filtering approach [47]. The filtering technique does not based on any induction algorithm. Filtered subset evaluation technique is scalable and fast. Figure 6 demonstrates the steps followed to find subset of feature by utilizing filter method.

5 Machine learning techniques

In the previous studies, number of frameworks were developed by implementing various machine learning algorithms like K-nearest Neighbour (KNN), Decision Tree (J48),

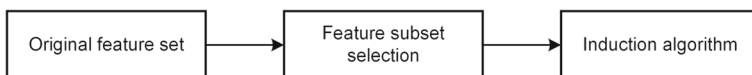


Fig. 6 Feature selection by utilizing filter approach

Naïve Bayes (NB), Multi-Layer Perceptron (MLP) and Random Forest (RF) by [81], Bayesian classification by [92], Bayes network, Multi-layer perceptron, J48, Random forest and K-Nearest neighbours by [63] and Support Vector Machine (SVM), Decision Tree (J48), Logistic Regression, Random Forest(RF), K-Nearest neighbors, Naïve Bayes(NB), Adaboost and Deep Learning by [30]. It is seen in the literature, that the performance of LSSVM [42, 48] in predicting the fault is better when compared to other machine learning algorithms. So, in this study, we use LSSVM classifier with three distinct kernel functions i.e., linear, RBF and polynomial which is described below:

5.1 LSSVM classifier

LSSVM being the part of supervised machine learning algorithm, is used in different fields for example : regression, classification and outliers detection [78]. The fundamental design of LSSVM machine learning algorithm is based on two-class problems, where data are divided on the basis of optimum hyperplane determined by support vectors. Boundary value of the training set is decided by the support vectors among two classes. In this research paper, LSSVM with distinct kernels being used as classifier for building a model to classify benign and malware apps. The comprehensive type of LSSVM function is determined as:

$$z(y) = a^T \phi(y) + b \quad (10)$$

here z is the output vector and y is the input vector, $\phi(y)$ is a non-linear mapped function and it is utilized to map the data input with greater size of feature space. a indicate the adapt weight vector and b represent the scalar threshold value accordingly. The succeeding equation is improved by the following manner:

$$\begin{aligned} & \text{Minimize } \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{x=1} I E_x^2 \\ & \text{subject to } y(x) = w^T \phi(x) + c + E_x, \quad x = 1, 2, 3, \dots, n, \end{aligned} \quad (11)$$

here E_x is the input measure by error sample a and γ be the cost function. By solving this problem, malware detection values are obtained from the equation given below:

$$\begin{aligned} Y' &= \sum_{x=1} I(\alpha - \alpha^*) \phi(b_i) * \phi(b) + c \\ &= \sum_{x=1} I(\alpha - \alpha^*) * K(b_i, b) + c \end{aligned} \quad (12)$$

here $K(b_i, b)$ is the function of kernel, in order to enable the product to be carried out in high-dimensional feature space by utilizing data space in low-dimension. The considered kernel functions in the present research paper are stated as follows:

1. Linear function:

$$K(x_i, x_j) = x_i^T x_j \quad (13)$$

2. Polynomial function:²⁸

$$K(x_i, x_j) = (x_i^T * x_j + C)^d \quad (14)$$

²⁸In our study, we fixed the value of T=3 and d=5 for performing experiment with both of the linear and polynomial kernel.

Table 6 Confusion matrix to classify a Android app is benign or malware (.apk)

	Benign	Malware
Benign	Benign-> Benign (TN)	Benign-> Malware (FP)
Malware	Malware-> Benign (FN)	Malware-> Malware (TP)

3. Radial basis function or RBF Kernel:²⁹

$$K(x_i, x_j) = e^{\gamma \|x_i - x_j\|}, \gamma > 0 \quad (15)$$

6 Comparison of proposed model with different existing techniques

To validate that our proposed framework is able to achieve higher detection rate or not, we compare the result of our proposed model with three different techniques mentioned below:

- a. **Comparison of results with previously used classifiers and frameworks:-** To validate that our proposed model is feasible to detect malware as equivalent to previous used classifiers or not, we calculate two performance parameters like Accuracy and F-measure for new proposed model and existing models. In addition to that, we compare our developed malware detection model with existing frameworks or approaches.
- b. **Comparison of results with different Anti-Virus scanners:-** To compare the performance of our model for malware detection, we chose ten available distinct anti-virus scanners and compare their detection rate with the detection rate of our proposed framework.
- c. **Detection of known and unknown malware families:-** Further, to evaluate how much our proposed framework is reliable to detect known and unknown malware families, we test known and unknown malware families with our proposed framework and calculate the accuracy to detect the malware.

7 Evaluation of performance parameters

In this section of the paper, we discuss the fundamental definitions of the performance parameters utilized by us while evaluating our proposed model for malware detection. Confusion matrix is used to calculate all these parameters. It consists of actual and detected classification information built by detection models. Table 6 demonstrates the confusion matrix for malware detection model. In the present work, two performance parameters namely, F-Measure and Accuracy are utilized for measuring the performance of malware detection approaches.

- True Positives (TP): A true positive is an outcome where the model correctly predicts the positive class.
- True Negative (TN) : A true negative is an outcome where the model correctly predicts the negative class.

²⁹In our study, we fixed the value of $\gamma = 10$ for performing experiment with RBF kernel.

- False Positive (FP): A false positive is an outcome where the model incorrectly predicts the positive class.
- False Negative (FN): A false negative is an outcome where the model incorrectly predicts the negative class.
- Precision: Precision quantifies the number of positive class predictions that actually belong to the positive class.

$$Precision = \frac{a}{a + b}, \quad (16)$$

- Recall: Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.

$$Recall = \frac{a}{a + c}, \quad (17)$$

where $a = N_{Malware \rightarrow Malware}$,
 $b = N_{Benign \rightarrow Malware}$,
 $c = N_{Malware \rightarrow Benign}$

Accuracy Accuracy is defined as corrected detection of malware-infected apps with total number of benign and malware-infected apps. For supervised, semi-supervised and hybrid machine learning techniques it is given by

$$Accuracy = \frac{a + d}{N_{classes}}, \quad (18)$$

where $N_{classes} = a + b + c + d$,
 $d = N_{Benign \rightarrow Benign}$

F-measure In this research paper, we develop distinct malware detection models by implementing distinct machine learning algorithms. So it is very difficult to compare two different models with high recall and low precision or vice versa. Therefore, in this study we used F-measure to compare two different models. F-measure helps to measure Precision and Recall at the same time. F-measure uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more and is defined by.

$$\begin{aligned} F - measure &= \frac{2 * Precision * Recall}{Precision + Recall} \\ &= \frac{2 * a}{2 * a + b + c} \end{aligned} \quad (19)$$

8 Experimental setup

In the present section, we introduce the experimental setup done to find the performance of our developed malware detection models. LSSVM with three distinct kernel functions (i.e., polynomial, RBF and linear) is implemented on thirty different categories of android apps mentioned in Table 3. All these data sets have varying number of benign or malware apps which are adequate to perform our analysis. Figure 7 shows the framework of our proposed model named as FSDroid.

In the very first step, feature ranking and feature subset selection approaches are applied on the extracted features data set. In the next step, we use the Min-max normalization

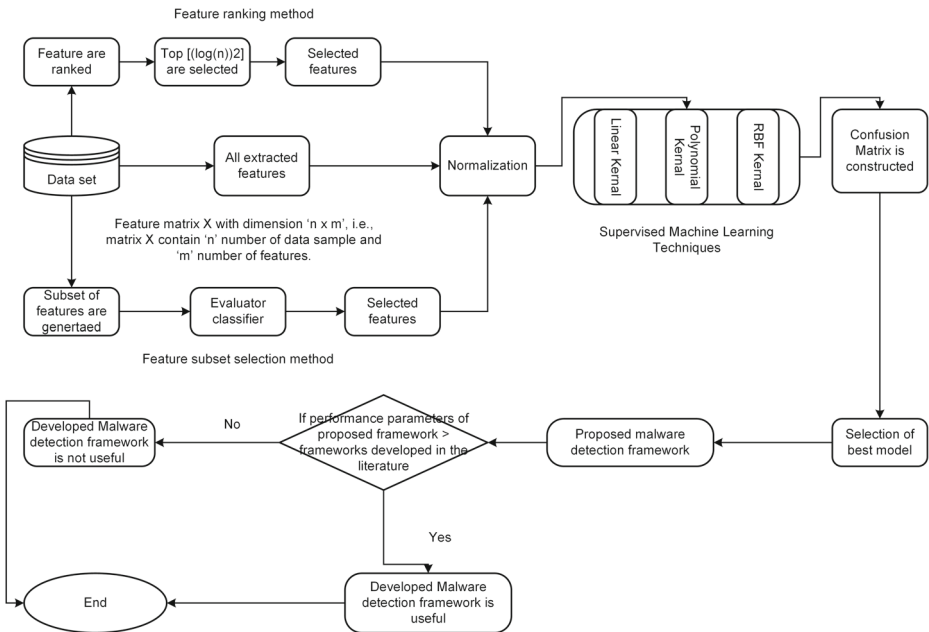


Fig. 7 Framework of FSDroid

approach to normalize the data. This approach is based on the principle of linear transformation, which bring each data point D_{q_i} of feature Q to a normalized value D_{q_i} , that lie in between $0 - 1$. Following equation is considered to find the normalized value of D_{q_i} :

$$Normalized(D_{q_i}) = \frac{D_{q_i} - \min(Q)}{\max(Q) - \min(Q)},$$

where $\min(Q)$ & $\max(Q)$ are the minimum and maximum significance of attribute Q , respectively. In the third step, we trained significant features by implementing distinct machine learning techniques. In the next step, we construct a confusion matrix and calculate the performance parameters i.e., accuracy and F-measure. Next, we compare the performance of the developed malware detection model and select the best malware detection model. At last, we compare the performance of our proposed malware detection model with existing techniques available in the literature and distinct anti-virus scanners. If the performance of our proposed malware detection model is better than existing techniques then it is useful and in reverse of it if the performance is not enhanced than the proposed malware model is not useful.

The subsequent measures are pursued at the time of selecting a subset of features to build the malware detection model that detects either the app is benign or malware. Feature selection approaches are implemented on 30 different data sets of Android apps. Hence, a total of 990 ((1 selecting all extracted features + 10 feature selection approaches) * 30 data sets (subsets of different feature sets particular to data sets determined after conducting feature selection) * 3 detection methods) different detection models have been build in this work. Below we provide step by step details of our approach:

1. In the present work, four feature subset selection approaches and six feature ranking approaches are implemented on 30 different feature sets to select the right set of features for malware detection.
2. The subsets of features obtained from aforementioned procedure are given as an input to machine learning classifiers. To compare the developed models, we use 20 fold cross-validation method. Cross-validation is a statistical learning approach that is utilized to classify and match the models by dividing the data into two different portions [47]. One portion is utilized to train and the remaining portion of data is utilized to verify the build model, on the basis of training [47]. The data is initially separated into K same sized segments. $K-1$ folds are utilized to train the model and the rest one fold is utilized for testing intention. K -fold cross-validation is having important significance in utilizing the data set for the both testing and training. For this study, 20-fold cross-validation is utilized to analyze the models, i.e., data sets are segregated into 20 portions. The outcomes of all build malware detection models are matched with each other by employing two distinct performance measure parameters: F-measure and Accuracy³⁰.
3. FSDroid i.e., proposed model build by utilizing above two steps are validated with the existing techniques developed in the literature to review whether the build malware detection model is useful or not.

9 Results of performed experiment

In the current section of the paper, the relationship among different feature sets and malware detection at the class level is submitted. Set of features are used as an input and present the ratio of benign and malware apps within an experiment. F-measure and Accuracy are used as performance assessment parameters to match the performance of malware detection model build by using LSSVM with distinct kernel functions (i.e., polynomial, RBF and linear). To depict the experimental results we utilize the abbreviations as given in Table 7 corresponding to their actual names.

9.1 Feature ranking approaches

Six feature ranking approaches: gain-ratio feature evaluation, Chi-squared test, information gain feature evaluation, logistic regression analysis, information gain, oneR feature evaluation and principal component analysis are implemented on a distinct feature sets. Each approach utilize distinct performance parameters to rank the feature. Moreover, top $\lceil \log_2 a \rceil$ set of features from “ a ” number of features being measured to build a model for detecting malware. For initial four feature ranking approaches (Gain-ratio feature evaluation, Chi-squared test, OneR feature evaluation and Information gain), top $\lceil \log_2 a \rceil$ are selected as subset of features, where a is the number of features in the original data set (for this work $a=20$). However, in the case of ULR, those features are selected which posses a positive value of regression co-efficient, i.e., p-value measure is below 0.05, and in matter of PCA, only those features are selected which have eigenvalue greater than 1. Considered features using feature ranking approaches are demonstrated in Fig. 8.

³⁰Performance Parameters are calculated on the basis of training and testing data set.

Table 7 Used naming convention in this study

Abbreviation	Corresponding name
DS	Data set
FS1	Correlation best Feature Selection
FS2	Classifier Subset Evaluation
FS3	Filtered Subset Evaluation
FS4	Rough Set Analysis (RSA)
FR1	Chi Squared test
FR2	Gain Ratio Feature Evaluation
FR3	Filtered Subset Evaluation
FR4	Information Gain Feature Evaluation
FR5	Logistic regression analysis
FR6	Principal Component Analysis (PCA)
AF	All Extracted features

9.2 Feature subset selection approaches

In the present work, four distinct kinds of feature subset selection approaches are implemented on thirty data sets of Android apps one after another. Feature subset selection approaches work on the principle of hypothesis which make models with better accuracy and make less amount of misclassified errors, while selecting the best features from available number of features. Later, these isolation subset of features has been selected as an input for building a model to detect either the app is benign or malware. Considered set of features after feature subset selection approaches are demonstrates in Fig. 9.

9.3 Machine learning techniques

Eleven subsets of features (1 considering all set of extracted features + 10 resulting by implemented feature selection approaches) are used as an input to build a model for malware detection. Hardware utilized to carry out this study is Core i7 processor having storage capacity of 1TB hard disk and 16GB RAM. Detection models are build by use the MATLAB environment. The performance of each detection model is measured by using two performance parameters: F-measure and Accuracy and in addition to this we also measured the time to build and test the model. Table 8, 9, 10, 11, 12 and 13, presents the performance values obtained for distinct data sets by utilizing LSSVM with linear, RBF and polynomial kernel function and time to build and test the models. On the basis of Table 8–13, it may be concluded that:

- Model developed by considering features selected by feature selection approaches as an input is able to detect malware more effectively rather than model developed by using all extracted feature sets.
- Model constructed by considering LSSVM with polynomial, RBF and linear kernel by selecting FS4 as an input achieved higher detection rate when compared to other models developed by using different feature selection approaches.

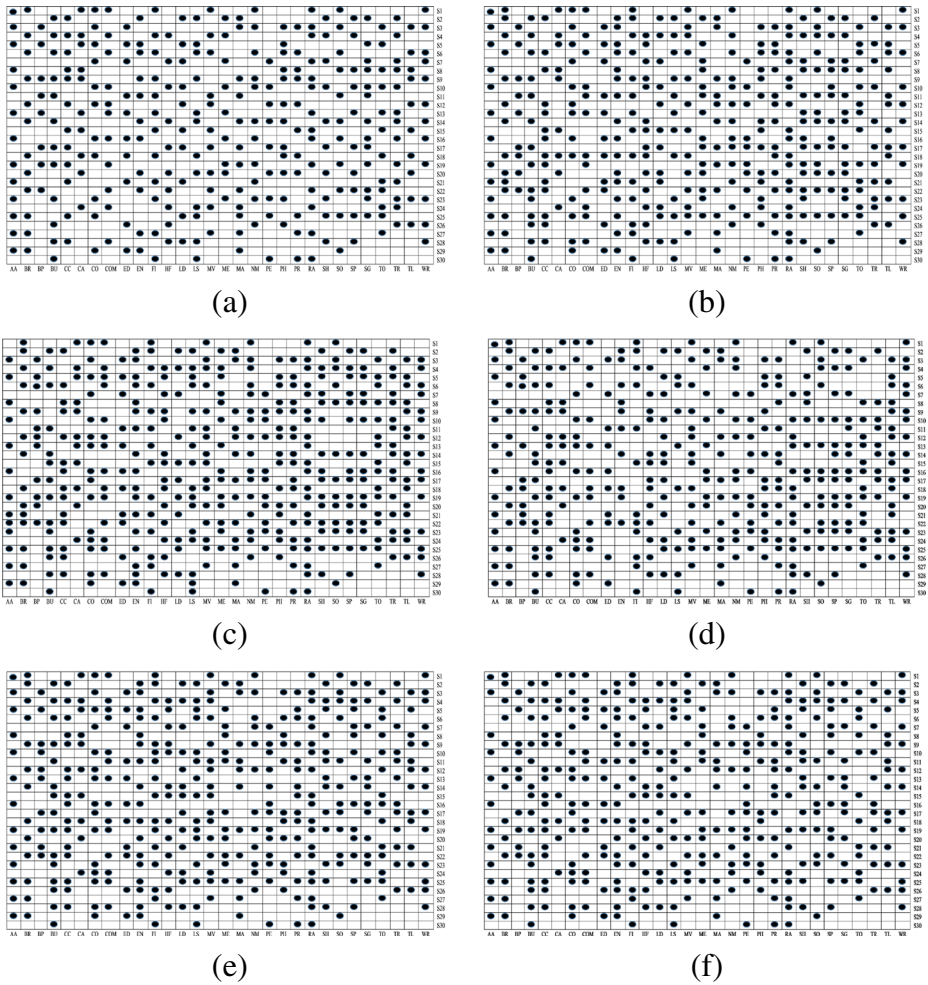


Fig. 8 Feature ranking methods

- Model build by considering LSSVM with RBF kernel by selecting FS4 as an input achieved higher detection rate when compared to other models developed by using polynomial and linear kernel.

In this research paper, LSSVM with three kernel functions and ten distinct feature selection approaches are considered to select features which helps to detect Android malware more effectively. To find out which developed model is more capable to detect malware, we construct box-plot diagrams of the individual model. Box-plot diagrams helps to identify which model is best suitable for malware detection on the basis of few number of outliers and better value of median. Figure 10a–f demonstrate the box-plot diagrams for F-measure and Accuracy for every developed model. The *x*-axis of the diagrams presents the feature

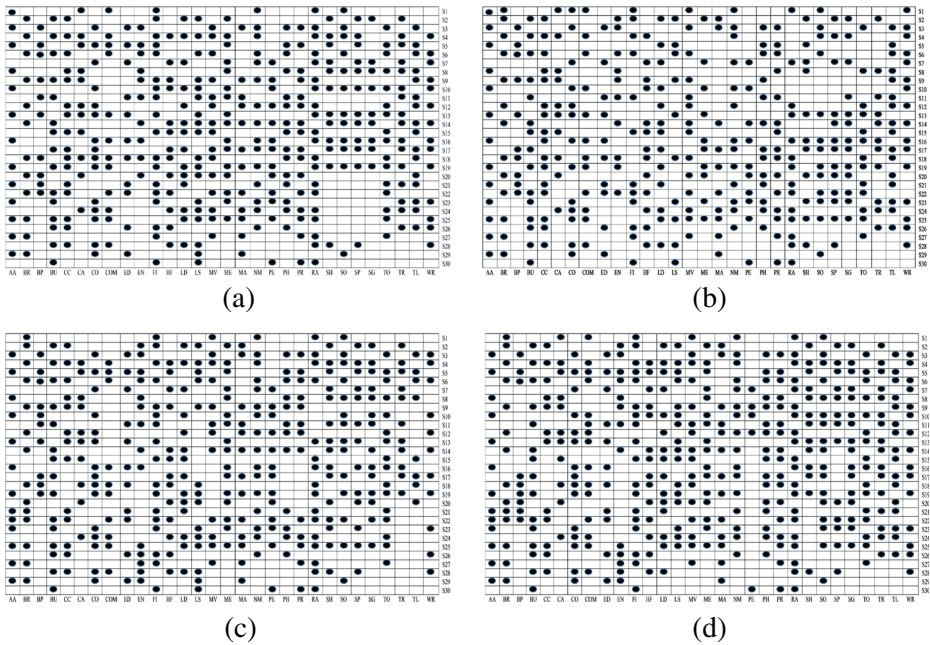


Fig. 9 Feature subset selection methods

selection techniques. Figures include eleven box-plot diagram: one box-plot diagram consists of all extracted feature sets, four box-plot consist of feature subset selection approaches and six box-plot consist of feature ranking approaches. On the basis of the box-plot diagram, we find following observations:

- Model constructed by considering *LSSVM with polynomial and linear kernel* and FS4 achieved higher median value in addition to few outliers. On the basis of box-plot diagrams demonstrated in Fig. 10a–f, model developed by considering FS4 as feature selection approach gives better detection rate when compared to other developed approaches.
- From box-plot diagrams, we observed that model build by considering *LSSVM with RBF kernel* and FS4, is having few outliers and higher median value. It means that the model developed by using FS4 for detecting malware and benign apps achieved better results when compare to other developed models in this study.
- In this study, we implement six distinct *feature ranking* approaches, to detect malware from Android apps. Among implemented feature ranking approaches, on the basis of box-plot diagrams we can conclude that, model developed by using FR6 achieved higher median value in addition with fewer outliers.
- On the basis of box-plot diagrams, among all implemented *feature subset selection* approaches, model developed by using FS4 give best results when compared to other approaches.

Table 8 Accuracy and F-measure measured by considering Linear kernel

ID	Accuracy														F-Measure													
	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4						
D1	72.33	83.33	85.0	87.67	87.66	88	87	88	86.6	87.6	88	0.88	0.9	0.89	0.89	0.9	0.85	0.88	0.9	0.86	0.86	0.9	0.9					
D2	78.8	80	82.08	86.27	82.66	88	86	83	86.6	87.6	93	0.80	0.88	0.82	0.84	0.83	0.82	0.86	0.87	0.86	0.86	0.92	0.92					
D3	76.8	82	84.8	84.67	81.06	83	85	82	81.6	85.6	94.7	0.82	0.81	0.80	0.81	0.80	0.81	0.82	0.81	0.85	0.86	0.90	0.90					
D4	72.8	82	81.08	82.27	81.60	86	85	82	85.6	89.6	92	0.78	0.83	0.80	0.82	0.81	0.80	0.82	0.89	0.88	0.87	0.89	0.89					
D5	70	76	81	82	86	82	93	81	86	87	90	0.70	0.81	0.88	0.89	0.86	0.87	0.91	0.86	0.85	0.82	0.88						
D6	78.8	80	82.08	86.27	82.66	88	86	83	86.6	87.6	93	0.80	0.88	0.82	0.84	0.83	0.82	0.86	0.87	0.86	0.86	0.92	0.92					
D7	67.8	72.8	82	85	87	88	88	81	80	87	96	0.78	0.81	0.80	0.82	0.81	0.80	0.82	0.83	0.88	0.89	0.91	0.91					
D8	67	79	72	78	81	80	92	82	83	86	90	0.71	0.81	0.80	0.82	0.80	0.80	0.89	0.83	0.83	0.84	0.85						
D9	80	86	88	93	86	95	97	97	92	92	95	0.88	0.89	0.92	0.94	0.93	0.92	0.97	0.99	0.91	0.92	0.91						
D10	66.8	78	81	87	86	82	81	88	89	89	91	0.78	0.82	0.81	0.88	0.86	0.87	0.85	0.88	0.82	0.88	0.96	0.96					
D11	79	88	88	86	86	89	89	83	86	89	92	0.78	0.87	0.86	0.86	0.85	0.87	0.85	0.84	0.82	0.85	0.96	0.96					
D12	77.8	82	88	82	86	81	83	86	86	86	90	0.81	0.82	0.80	0.82	0.81	0.81	0.82	0.86	0.86	0.84	0.91	0.91					
D13	69.8	78	81	81	82	82	81	82	86	86	89	0.61	0.78	0.80	0.81	0.80	0.79	0.81	0.82	0.80	0.81	0.88	0.88					
D14	77	81	80	86	82	85	82	81	86	86	90	0.77	0.82	0.86	0.81	0.80	0.88	0.83	0.85	0.81	0.82	0.89	0.89					
D15	72	88	88	86	86	88	93	89	91	90	92	0.78	0.86	0.88	0.83	0.86	0.85	0.90	0.84	0.86	0.86	0.89	0.89					
D16	67	71	80	87	86	82	83	81	86	87	88	0.72	0.82	0.81	0.80	0.80	0.81	0.83	0.80	0.80	0.81	0.83	0.83					
D17	80	86	88	86	89	90	92	93	96	91	98	0.81	0.88	0.81	0.86	0.88	0.88	0.89	0.89	0.89	0.88	1	1					
D18	82	89	89	91	90	91.8	92	92.9	95	96	99	0.80	0.89	0.85	0.85	0.87	0.84	0.85	0.86	0.88	0.87	0.93	0.93					
D19	81	86	87	89	91	92	98	94	95	96	96	0.82	0.89	0.86	0.87	0.88	0.89	0.96	0.92	0.90	0.88	0.97	0.97					
D20	78	82	86	86	89	90	91	93	96	91	90	0.78	0.82	0.86	0.83	0.82	0.85	0.87	0.88	0.89	0.86	0.87	0.87					
D21	67	79	78	80	82	87	86.7	89	90	89	91.7	0.71	0.82	0.85	0.84	0.85	0.87	0.88	0.838	0.87	0.89	0.93	0.93					
D22	78	88	86	87	89.7	89.3	98	92	94	95	95	0.77	0.82	0.88	0.87	0.85	0.82	0.95	0.85	0.84	0.86	0.90	0.90					
D23	68.9	80	81	80.88	83.76	88.78	89.71	89.9	92	90	90.1	0.688	0.72	0.809	0.846	0.80	0.820	0.85	0.84	0.82	0.80	0.81	0.81					
D24	65	81	80	80.8	82	89	92.2	91.3	90	89.7	92	0.70	0.80	0.81	0.82	0.86	0.85	0.89	0.82	0.82	0.85	0.85	0.85					
D25	80	86	88	86	89	90	92	93	96	91	98	0.81	0.88	0.81	0.86	0.88	0.88	0.89	0.89	0.89	0.88	1	1					
D26	76	84	88.7	86.9	89.6	90.8	92.1	94	97	95	98	0.78	0.82	0.83	0.84	0.85	0.86	0.86	0.85	0.88	0.88	0.92	0.92					
D27	67	82	85	86	81	90.1	91.9	93.6	97	96.9	97.9	0.71	0.82	0.86	0.88	0.89	0.89	0.87	0.85	0.86	0.87	0.91	0.91					
D28	67	86	88	86	89	90	95	93	86	81	89	0.71	0.80	0.82	0.85	0.82	0.85	0.89	0.82	0.88	0.85	0.88	0.88					
D29	67	78	82	85	86	89	91	90	85	84	89	0.71	0.80	0.81	0.84	0.86	0.88	0.89	0.86	0.87	0.85	0.88	0.88					
D30	60	76	82	84	87	89	91	92	95	91	98	0.671	0.72	0.81	0.82	0.85	0.86	0.87	0.85	0.87	0.85	0.87	0.87	0.87				

Table 9 Accuracy and F-measure measured by considering Polynomial kernel

ID	AF	FRI	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AF	FRI	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
D1	62.33	80.33	81.0	80.67	80	80.9	82.66	80.8	80.6	80.96	82	0.78	0.80	0.89	0.89	0.85	0.88	0.9	0.89	0.87	0.86	0.85
D2	64.33	82.33	84.0	82.67	82	82.9	84.6	82.8	81.6	82.96	83	0.72	0.80	0.81	0.86	0.88	0.81	0.82	0.84	0.82	0.81	0.85
D3	62.33	70.83	71.0	79.6	78	76.9	72.08	70.56	70.6	78.2	89.06	0.68	0.78	0.79	0.79	0.79	0.78	0.72	0.77	0.76	0.75	0.80
D4	73.33	78.33	80.9	80.97	82	83.9	81.8	82.6	86.6	85	91.01	0.68	0.78	0.79	0.83	0.84	0.87	0.82	0.87	0.86	0.85	0.89
D5	71.03	79.33	81.0	82.67	82	82.9	92.0	81.8	80.56	81.96	80	0.68	0.78	0.85	0.83	0.82	0.85	0.89	0.80	0.82	0.83	0.80
D6	72.33	78.33	79	80.7	81	72.9	78	86.8	85.96	85	93	0.68	0.78	0.81	0.82	0.82	0.86	0.89	0.81	0.82	0.79	0.90
D7	71	83	82	82	84	85	93	82	84	86	81	0.70	0.82	0.83	0.80	0.82	0.86	0.88	0.83	0.82	0.84	0.81
D8	75.03	78	82	87	88	89	88	86	86	88	92	0.78	0.82	0.89	0.88	0.82	0.86	0.89	0.82	0.86	0.89	0.9
D9	73.8	83	86	87	88.77	86.79	98	85.80	84.5	82.6	88.81	0.68	0.78	0.85	0.86	0.88	0.87	0.92	0.82	0.85	0.82	0.87
D10	75	87	88	86	90	91.9	90.8	90.6	90.6	92	98	0.78	0.82	0.89	0.89	0.87	0.86	0.89	0.86	0.86	0.82	0.9
D11	68.03	78.33	86.0	87	91	89	98.9	88	88	89	92	0.79	0.82	0.87	0.88	0.90	0.88	0.95	0.82	0.86	0.83	0.85
D12	62.33	78.81	80.8	87	82	83	86	85	83	87	92	0.78	0.82	0.87	0.88	0.82	0.86	0.80	0.83	0.85	0.83	0.89
D13	72.3	86.31	86.8	88.7	89	89	87	83	86	92	98.7	0.69	0.82	0.86	0.87	0.81	0.82	0.81	0.82	0.85	0.81	0.89
D14	72.1	83.83	85.90	91.05	89	88	97.7	86	85	86	88	0.78	0.80	0.86	0.82	0.82	0.83	0.88	0.81	0.85	0.84	0.83
D15	72.88	86.93	88.90	89.07	88	89	98.89	88	86	86	89	0.78	0.82	0.87	0.87	0.82	0.85	0.89	0.82	0.85	0.83	0.82
D16	67.8	81.33	85.0	86.7	88	90.9	91.8	92.6	93.6	92	99.6	0.68	0.81	0.83	0.88	0.82	0.86	0.81	0.85	0.84	0.85	0.94
D17	78.23	87.33	85.0	88.71	89	92	99.2	93	94	95	92	0.78	0.82	0.88	0.81	0.88	0.89	1	0.9	0.82	0.85	0.89
D18	72	83	80	81	82	83	85	84	86	83	92	0.68	0.81	0.82	0.83	0.81	0.83	0.82	0.84	0.82	0.81	0.89
D19	62.3	78.3	81.07	83.67	88	87.9	97.06	86.8	86.6	85.6	89.8	0.68	0.82	0.83	0.88	0.82	0.83	0.89	0.81	0.83	0.85	0.82
D20	68.3	82.33	85.8	86.7	82	83.7	88.8	89.2	88.67	87.7	98.86	0.67	0.81	0.85	0.82	0.83	0.83	0.81	0.83	0.83	0.85	0.87
D21	72.33	87	88	89	89	80.7	98	88	86	86	88	0.79	0.82	0.86	0.83	0.82	0.81	0.89	0.80	0.83	0.85	0.83
D22	62	78.83	81.0	85	88	89	85	86	92	90	98	0.66	0.86	0.80	0.82	0.81	0.82	0.81	0.83	0.83	0.80	0.88
D23	71	78.1	82	87	82	81	92	81	86	86	83	0.68	0.78	0.82	0.83	0.81	0.82	0.89	0.81	0.82	0.83	0.85
D24	61	82	87	85	86	83	83	85	84	81	98.91	0.62	0.80	0.82	0.83	0.82	0.80	0.85	0.85	0.83	0.84	0.89
D25	70	88.1	88	89	92	91.99	87	87	88	85	0.69	0.72	0.81	0.84	0.82	0.83	0.88	0.80	0.83	0.84	0.81	
D26	70	82.1	84.77	85.0	88	87	97	85	82	85	88	0.62	0.81	0.80	0.84	0.82	0.85	0.87	0.86	0.86	0.87	0.86
D27	69	87.9	86	87.3	86	88	87	86.9	86.2	83.8	97	0.68	0.81	0.86	0.87	0.83	0.86	0.87	0.88	0.86	0.87	1
D28	69	72.9	78	80	81	83	83	85	84.9	83.8	95	0.62	0.78	0.81	0.82	0.82	0.81	0.80	0.81	0.82	0.83	0.84
D29	67	87.9	88.7	89.8	87	88	91	96	92	95	98	0.62	0.81	0.86	0.85	0.85	0.87	0.88	0.89	0.87	0.86	0.95
D30	69	81	82.8	89.7	81	80	82	89	88	87	90	0.78	0.81	0.85	0.87	0.82	0.83	0.85	0.86	0.86	0.87	0.90

Table 10 Accuracy and F-measure measured by considering RBF kernel

Accuracy		F-Measure																				
ID	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
D1	78.33	82	84	86	86	82	83	85	86	83	89	0.79	0.81	0.85	0.83	0.81	0.83	0.85	0.82	0.89	0.81	0.91
D2	72	80.88	84	87	86	85	82	85	86	88	91	0.75	0.82	0.86	0.85	0.84	0.81	0.85	0.83	0.85	0.81	0.87
D3	67	81	84	87	82	85	82	81	84	89	90	0.78	0.87	0.86	0.85	0.83	0.85	0.86	0.85	0.84	0.87	0.89
D4	72.8	88	86	89	86	83	89	85	87	89	90	0.72	0.80	0.88	0.84	0.87	0.86	0.86	0.87	0.81	0.86	0.88
D5	73	81	83	80	81	88	88	82	83	84	89	0.67	0.80	0.81	0.82	0.83	0.83	0.84	0.85	0.86	0.87	0.93
D6	67	85	87	86	85	84	88	89	91	93	96	0.69	0.88	0.85	0.86	0.87	0.87	0.85	0.88	0.89	0.89	0.96
D7	72	81	85	88	89	89.6	88.7	86.8	89.7	91	95	0.70	0.86	0.85	0.84	0.87	0.89	0.86	0.87	0.89	0.81	0.90
D8	65	78	75	78	82	84	85	86	87	88	91	0.67	0.81	0.81	0.88	0.85	0.84	0.83	0.84	0.84	0.88	0.89
D9	68	84	87	92	91	83	97	84	95	93	86	0.78	0.89	0.92	0.94	0.93	0.92	0.99	0.96	0.91	0.92	0.91
D10	66.8	78	86	88	89	82	89	89	89.8	89.7	97	0.70	0.82	0.81	0.88	0.86	0.87	0.85	0.88	0.82	0.88	0.96
D11	79	88	88	86	86	89	98	89	80	86	88	0.72	0.87	0.86	0.86	0.85	0.87	0.93	0.85	0.84	0.82	0.85
D12	66.8	82	88	82	86	81	83	88	87	89	90	0.75	0.80	0.81	0.82	0.81	0.81	0.82	0.86	0.86	0.84	0.89
D13	69.1	78	81	81	82	82	87	82	86	88	89	0.60	0.78	0.80	0.81	0.80	0.79	0.81	0.82	0.80	0.81	0.88
D14	67	81	80	86	82	85	90.9	82	81	86	86	0.67	0.82	0.86	0.81	0.80	0.88	0.89	0.83	0.85	0.81	0.82
D15	69.7	88	88	86	86	88	96	88	89.8	91	92	0.69	0.86	0.88	0.83	0.86	0.85	0.94	0.83	0.84	0.86	0.86
D16	67	71	80	87	86	82	83	81	86	87	88	0.72	0.82	0.81	0.80	0.80	0.81	0.83	0.80	0.80	0.81	0.83
D17	80	86	88	86	89	90	92	93	96	91	98	0.67	0.86	0.82	0.85	0.87	0.88	0.82	0.85	0.88	0.98	1
D18	72	87	89	91	90	92.8	91	92.9	95	96	99	0.70	0.89	0.85	0.85	0.87	0.84	0.85	0.86	0.88	0.90	0.93
D19	77	86	87	89	91	92	93	92	95	96	98	0.72	0.89	0.86	0.87	0.88	0.92	0.91	0.92	0.90	0.88	0.95
D20	68	88	86	86	89	90	95	92	93	91	92	0.78	0.82	0.86	0.83	0.82	0.85	0.89	0.87	0.88	0.84	0.85
D21	62	78	78	80	81	80	85.7	80.7	82	83	84	0.67	0.82	0.85	0.84	0.85	0.87	0.9	0.88	0.88	0.87	0.89
D22	69.8	83	85	87	88	88	90	92	96	95	98	0.68	0.82	0.88	0.87	0.85	0.82	0.85	0.85	0.88	0.89	0.91
D23	68.9	80	81	80.88	83.76	87.78	87.71	87.9	91	90	90	0.68	0.72	0.809	0.846	0.80	0.82	0.85	0.83	0.82	0.80	0.82
D24	65	81	80	82	83	89	90.3	89.2	90	89.7	88	0.67	0.80	0.81	0.82	0.86	0.85	0.89	0.82	0.82	0.85	0.85
D25	69.9	86	88	86	89	97	92	93	96	97	98	0.77	0.88	0.81	0.86	0.88	0.88	0.89	0.89	0.89	0.88	1
D26	69.9	84	88.7	86.9	89.6	94.8	96.1	94	97	95	97	0.73	0.82	0.83	0.84	0.85	0.86	0.86	0.85	0.88	0.86	0.92
D27	67	82	85	86	81	90.1	91.9	93.6	97	95.8	97.9	0.71	0.82	0.86	0.88	0.89	0.89	0.87	0.85	0.86	0.87	0.91
D28	63	82	86	89	86	92	96	91	86	81	91	0.78	0.78	0.82	0.85	0.82	0.85	0.89	0.86	0.88	0.85	0.88
D29	67	78	82	85	86	82	92	87	85	84	89	0.77	0.80	0.81	0.84	0.86	0.88	0.87	0.87	0.86	0.85	0.88
D30	60	76	82	84	87	89	91	92	95	91	97	0.67	0.72	0.71	0.82	0.85	0.86	0.87	0.85	0.87	0.87	1

Table 11 Time to develop and test the malware detection models using Linear Kernel (in seconds)

Measured time											
ID	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
D1	720	480	360	360	360	300	280	320	300	295	250
D2	900	580	460	460	460	400	380	420	400	395	350
D3	600	480	360	360	360	300	280	220	200	295	200
D4	900	580	460	460	460	400	380	420	400	395	350
D5	800	480	420	430	410	405	280	320	300	395	350
D6	600	380	360	360	360	300	300	320	300	295	250
D7	650	350	320	320	320	280	280	220	290	295	200
D8	600	380	360	360	360	300	200	320	300	295	250
D9	600	380	360	360	360	300	200	320	300	295	250
D10	650	380	360	360	360	300	300	320	300	295	210
D11	700	380	360	360	360	300	300	320	300	295	200
D12	710	380	360	360	360	300	300	320	300	295	200
D13	710	380	360	360	360	300	300	320	300	295	210
D14	730	480	460	460	360	300	300	320	320	295	210
D15	750	420	410	410	360	300	200	220	220	205	205
D16	750	420	410	410	360	300	205	220	220	205	200
D17	650	320	310	310	330	300	200	210	210	200	180
D18	630	320	310	310	330	300	202	210	210	202	160
D19	620	320	310	310	330	300	180	210	210	200	190
D20	620	320	330	310	310	300	190	210	170	200	180
D21	660	310	320	320	330	300	200	210	210	200	170
D22	610	320	310	310	330	300	200	210	210	200	210
D23	650	310	320	320	330	300	200	210	190	200	210
D24	650	320	310	310	330	300	190	210	210	200	200
D25	600	310	300	300	310	300	200	210	210	200	160
D26	610	300	310	320	330	300	200	210	210	200	170
D27	650	320	310	310	330	300	200	210	210	200	180
D28	550	320	310	310	330	300	180	210	210	200	190
D29	580	320	310	310	330	300	180	220	200	210	190
D30	530	320	300	300	330	300	180	210	210	200	160

9.4 Comparison of results

To identify that out of implemented feature selection approaches and machine learning algorithms which technique work well or all of the techniques perform equally well, we employed pair-wise *t*-test in our study.

1. *Feature Selection Approaches*: In this study, for each of the feature selection approaches two sets are formed, each of feature selection approach have 90 distinct data points (3 machine learning techniques * 30 data set). *t*-test are performed on distinct feature selection approaches and the respective p-value to measure its statistical significance is compared. The outcome of *t*-test study is demonstrated in Fig. 11a. In the figure, we

Table 12 Time to develop and test the malware detection models using Polynomial Kernel (in seconds)

Measured Time											
ID	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
D1	720	480	360	360	360	300	290	320	300	295	295
D2	900	580	460	460	460	400	380	420	400	395	398
D3	600	480	360	360	360	300	280	220	200	295	190
D4	920	580	460	460	460	400	380	420	400	395	320
D5	780	480	420	430	410	405	290	320	300	395	300
D6	650	380	360	360	360	300	300	320	300	295	250
D7	680	350	320	320	320	280	210	220	290	295	260
D8	600	380	360	360	360	300	280	320	300	295	240
D9	600	380	360	360	360	300	200	320	300	295	250
D10	650	380	360	360	360	300	300	320	300	295	230
D11	720	380	360	360	360	300	280	320	300	295	298
D12	710	380	360	360	360	300	300	320	300	295	290
D13	730	380	360	360	360	300	300	320	300	295	280
D14	730	480	460	460	360	300	294	320	320	295	297
D15	750	420	410	410	360	300	198	220	220	205	205
D16	750	420	410	410	360	300	205	220	220	205	200
D17	650	320	310	310	330	300	200	210	210	200	280
D18	630	320	310	310	330	300	202	210	210	202	200
D19	620	320	310	310	330	300	190	210	210	200	190
D20	620	320	330	310	310	300	190	210	270	200	180
D21	660	310	320	320	330	300	190	210	210	200	200
D22	610	320	310	310	330	300	200	210	210	200	205
D23	650	310	320	320	330	300	200	210	195	200	210
D24	650	320	310	310	330	300	200	210	210	208	209
D25	600	310	300	300	310	300	200	210	210	200	190
D26	610	300	310	320	330	300	200	210	210	200	230
D27	650	320	310	310	330	300	200	210	210	200	180
D28	550	320	310	310	330	300	200	210	210	200	198
D29	580	320	310	310	330	300	200	220	200	210	190
D30	530	320	300	300	330	300	180	210	210	200	160

used two different symbols to represent the p-value i.e., circle filled with green color have p-value > 0.05 (having no relevance difference) and circle filled with red color have p-value ≤ 0.05 (relevance difference). After observing the Fig. 11a, it is clear that, majority of the cells are filled with green color circle. This means that there is no relevance difference among the employed feature selection approaches. Further, by determining the measure of mean difference, given in Table 14, we have observed that feature sets obtained by considering FS4 give best outcomes when examined with other implemented feature selection approaches.

Table 13 Time to develop and test the malware detection models using RBF Kernel (in seconds)

Measured Time											
ID	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
D1	720	480	360	360	360	300	280	320	300	295	230
D2	900	580	460	460	460	400	380	420	400	395	320
D3	600	480	360	360	360	300	280	220	200	295	190
D4	920	580	460	460	460	400	380	420	400	395	330
D5	780	480	420	430	410	405	280	320	300	395	230
D6	650	380	360	360	360	300	300	320	300	295	250
D7	680	350	320	320	320	280	280	220	290	295	200
D8	600	380	360	360	360	300	280	320	300	295	240
D9	600	380	360	360	360	300	190	320	300	295	250
D10	650	380	360	360	360	300	300	320	300	295	210
D11	720	380	360	360	360	300	280	320	300	295	298
D12	710	380	360	360	360	300	300	320	300	295	210
D13	730	380	360	360	360	300	300	320	300	295	220
D14	730	480	460	460	360	300	290	320	320	295	297
D15	750	420	410	410	360	300	190	220	220	205	205
D16	750	420	410	410	360	300	205	220	220	205	200
D17	650	320	310	310	330	300	200	210	210	200	180
D18	630	320	310	310	330	300	202	210	210	202	160
D19	620	320	310	310	330	300	180	210	210	200	170
D20	620	320	330	310	310	300	190	210	270	200	180
D21	660	310	320	320	330	300	190	210	210	200	200
D22	610	320	310	310	330	300	200	210	210	200	190
D23	650	310	320	320	330	300	200	210	195	200	210
D24	650	320	310	310	330	300	190	210	210	200	200
D25	600	310	300	300	310	300	200	210	210	200	160
D26	610	300	310	320	330	300	200	210	210	200	170
D27	650	320	310	310	330	300	200	210	210	200	180
D28	550	320	310	310	330	300	180	210	210	200	190
D29	580	320	310	310	330	300	170	220	200	210	190
D30	530	320	300	300	330	300	180	210	210	200	160

In the present work, we also compare the developed model on the basis of cost-benefit analysis. For every feature selection approach, cost-benefit analysis is computed by employing following equation:

$$Cost - Benefit = (Based_{cost} + Benefit_{cost})/2. \quad (20)$$

Here, $Based_{cost}$ is dependent on the correlation among the selected features set and error in the class. $Based_{cost}$ can be calculated from the following equation :

$$Based_{cost} = Accuracy (SM) * \rho_{SM.fault}. \quad (21)$$

Here, $Accuracy (SM)$ is the classification accuracy to build a malware detection model by utilizing selected features set, $\rho_{SM.fault}$ is a multiple correlation coefficient among

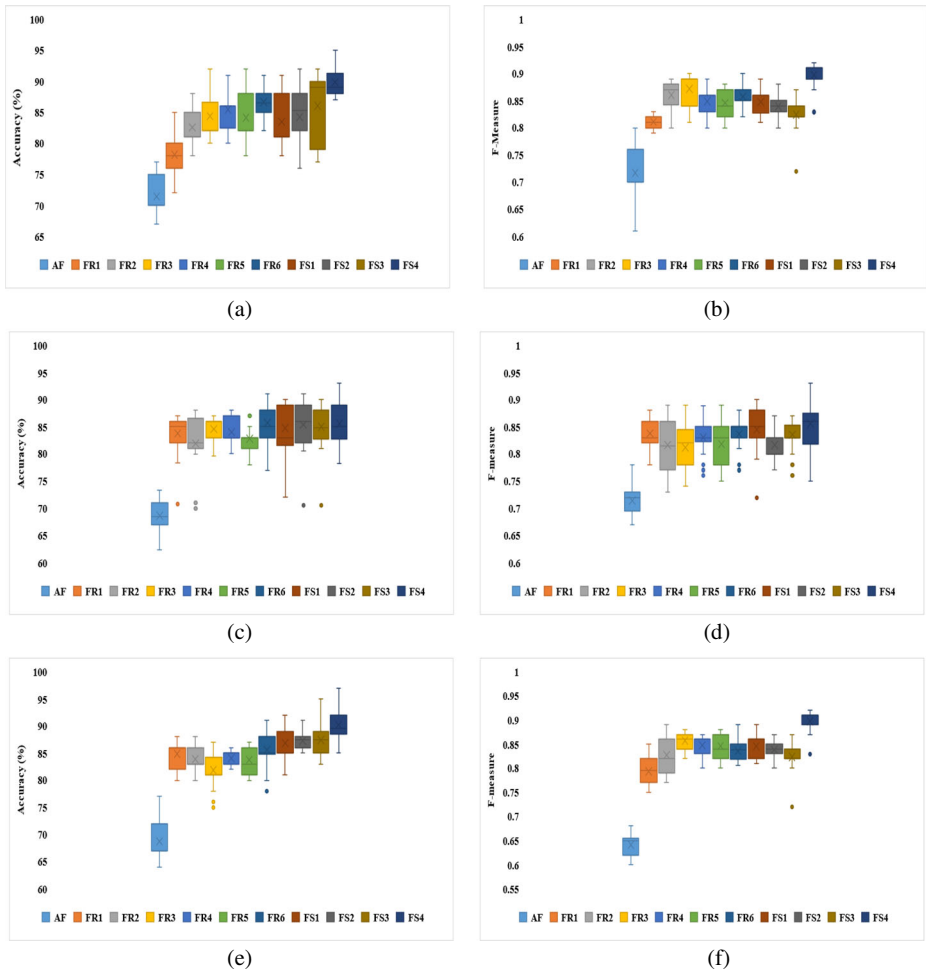


Fig. 10 Box-plot diagram of F-measure and Accuracy

selected features set and error. The proposed model produces higher accuracy and as it have higher multiple correlation coefficient so it will achieve a high $Based_{cost}$. NAM is considered as feature sets and NSM is considered as the number of selected features after implementing features selection approaches. $Based_{cost}$ can be calculated from the following equation:

$$Based_{cost} = NAM - NSM / NAM \tag{22}$$

The feature selection approach which achieve higher value of cost-benefit is an foremost feature selection approach as proposed in [16]. Figure 12a-b demonstrates cost-benefit of distinct feature selection approaches. On the basis of Fig. 12a-b we observed that FS4 achieved higher median Cost-benefit measure when matched with other approaches.

2. *Machine Learning Techniques*: In our study, we implemented eleven different features subsets (i.e., 1 considering all features + 10 feature selection approaches) on thirty different Android app data set by examining two performance parameters i.e., F-measure

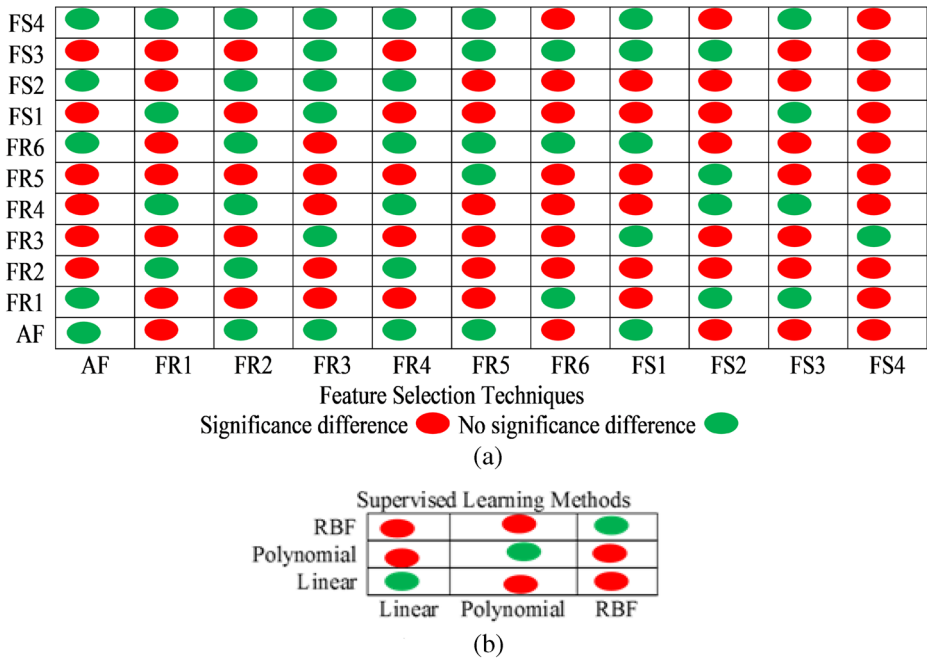


Fig. 11 a Feature selection techniques. b Classification methods

and Accuracy, all with 330 data points ((1 considering all set of features + 10 feature selection method) * 30 data sets)). Figure 11b demonstrates the outcomes of *t*-test analysis. On the basis of Fig. 11b, it is noticeable that, there is a relevance difference among these techniques because p-value is smaller than 0.05. On the other hand, by determining the difference in their mean value as given in Table 15, LSSVM with RBF kernel gives best outcome when compared to other machine learning techniques.

Table 14 Performance of distinct feature selection approaches after calculate its mean difference

Accuracy											
	AF	FR1	FR2	FR3	FR4	FR5	FR6	FS1	FS2	FS3	FS4
AF	0	-1.9	-0.96	-0.78	-1.91	-1.90	-4.89	-1.77	-1.80	-0.87	-5.8
FR1	1.8	0	0.77	0.87	-0.78	-0.80	-3.8	0.07	0.32	0.80	-3.89
FR2	0.87	-0.78	0	0.5	-2.0	-2.0	-3.89	-0.9	-0.32	0.20	-4.54
FR3	0.67	-0.68	-0.2	0	-1.32	-1.32	-4.08	-0.8	-0.45	0.07	-4.88
FR4	1.88	0.77	1.22	1.36	0	0	-2.99	0.77	0.8	1.7	-3.66
FR5	1.88	0.77	1.22	1.36	0	0	-2.99	0.75	0.8	1.7	-3.66
FR6	4.5	3.88	3.22	4.09	2.88	2.88	0	3.55	3.67	4.19	-0.50
FS1	1.65	-0.09	0.61	0.77	-0.80	-0.81	-3.88	0	0.21	0.80	-3.98
FS2	1.09	-0.29	0.39	0.51	-0.9	-0.9	-3.81	-0.22	0	0.8	-4.21
FS3	0.87	-0.88	-0.21	-0.09	-1.8	-1.8	-3.8	-0.08	-0.7	0	-4.89
FS4	6.0	3.9	4.89	4.88	3.88	3.88	0.48	3.88	4.16	4.77	0

Table 15 Performance of different supervised methods after calculate its mean difference

	Accuracy		
	Linear	Polynomial	RBF
Linear	0	−3.2	−5.98
Polynomial	2.86	0	−3.81
RBF	5.96	3.87	0

3. *Feature subset selection and feature ranking approaches*: For this study, pair-wise *t*-test is used to identify which feature selection approach work better. For both of the implemented approaches (i.e., feature subset selection and feature ranking) sample pairs of performance evaluation are studied. The performance of averaged feature subset selection and feature ranking techniques outcomes of *t*-test analysis are briefed in Table 16. In this research paper, three distinct kinds of machine learning algorithms are applied on thirty different Android categories by selecting Accuracy and F-measure as performance parameters, in accordance with each feature selection approaches an aggregate number of two sets are utilized, feature subset selection with 360 distinct points (which means 4 feature subset selection approaches * 3 machine learning techniques * 30 data sets) and feature ranking with 540 distinct data points (3 machine learning techniques * 6 feature ranking approaches * 30 data sets). On the basis of Table 16, it is seen that, there isn't a relevant variation among two implemented approaches, because p-value come out to be greater than 0.05. By comparing the value of the mean difference, feature subset selection approaches give best results as compared to feature ranking approaches. On the basis of *Cost-Benefit* analysis as demonstrated in Fig. 12, we can say that both feature subset selection and feature ranking have nearly similar *Cost-Benefit* value. It proves that the averaged cost and benefit of model build by considering selected set of features with feature subset selection approaches and feature ranking having nearly same value.

9.5 Evaluation of FSDroid

9.5.1 Comparison of results with previously used classifiers and frameworks

In addition to the study done in finding the best approach to build a malware detection model accurately, this study also makes the comparison with different most often used supervised machine learning approaches present in literature such as SVM with three distinct kernels i.e., linear, polynomial and RBF, Decision tree analysis, Logistic regression, Neural network and Naïve Bayes classifier. Figure 13 demonstrates the box-plot diagrams for F-measure and

Table 16 *t*-test analysis among feature subset selection approaches and feature ranking approaches

Accuracy	p-value	t-value
Mean(FR-FS)		
−0.1908	0.899	−0.3211
F-Measure		
−0.0078	0.599	−0.5251

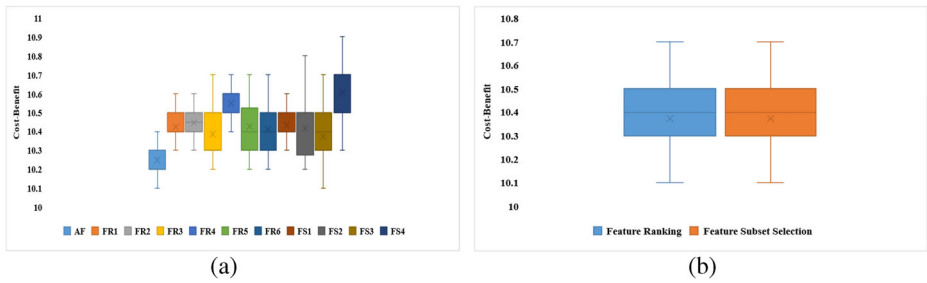


Fig. 12 Cost-benefit value

Accuracy of commonly utilized classifiers and LSSVM using linear, RBF and polynomial kernel. On the basis of Fig. 13, we observed that LSSVM using three distinct kernels have higher median value along with some number of outliers.

Pair-wise *t*-test is also implemented to decide which machine learning approach yield best performance. The outcomes of *t*-test study for distinct machine learning approaches are demonstrated in Fig. 14. On the basis of Fig. 14, it is seen that in number of the cases there is a relevance difference among these machine learning techniques because p-value is smaller than 0.05. On the other hand by noticing the mean difference value in Table 17 it can be seen that LSSVM with RBF kernel achieved better results when compared to other supervised machine learning techniques.

In addition to that, in our study we compare our proposed malware detection model (i.e., FSDroid) with existing frameworks or approaches that were developed in the literature. Table 18 shows the name, goal, methodology, deployment, data set and detection rate of suggested approaches or frameworks. Experiment was performed with our collected data set and empirical result reveals that our proposed framework has achieved 3% higher detection rate when compared to distinct framework available in the literature.

9.5.2 Comparison of results with different Anti-Virus scanners

Although our proposed framework LSSVM with RBF kernel i.e., FSDroid gives a better detection rate when compared to the machine learning technique used in the literature,

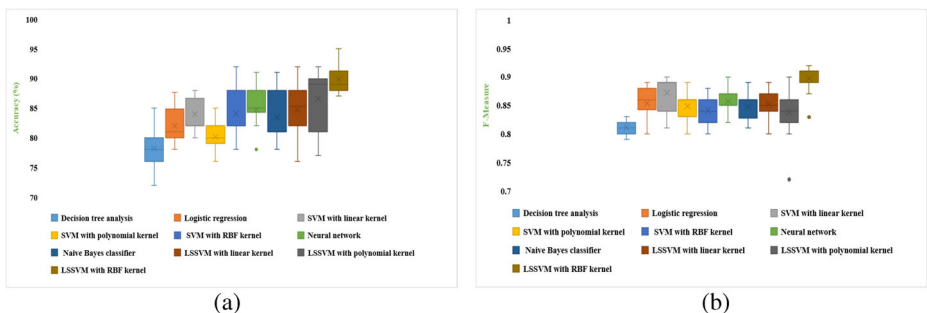


Fig. 13 Diagram of box-plot showing performance of different classifiers

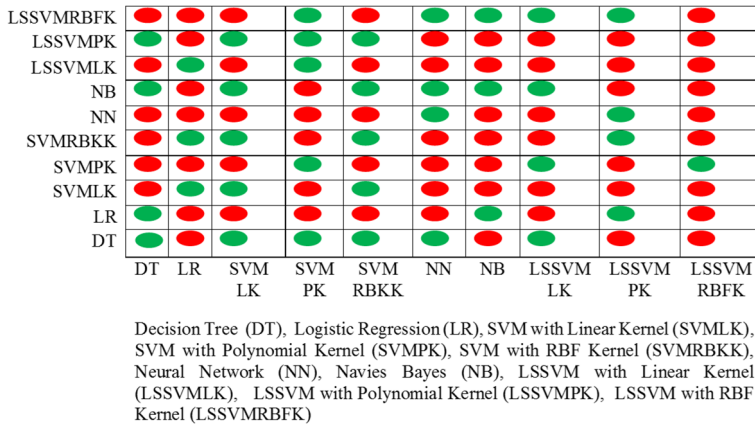


Fig. 14 *t*-test analysis (p-value)

in the end it must be comparable with the common anti-virus products available in practice for Android malware detection. For this experiment, we select 10 different anti-viruses which are available in the market and applied them on our collected data set.³¹ The performance of our proposed framework is comparatively better when compared to different anti-viruses available in the market. Table 19 shows the results of the experiment with anti-virus scanners. The detection rate of the anti-viruses scanners varies considerably. Also the best anti-virus scanners detected 97.1% of the Android malwares and certain scanners identified only 82% of the malicious samples, likely do not being specialized in detecting Android malware. Our proposed framework i.e., FSDroid gives the detection rate of 98.8% and outperforms 1 out of 10 anti-virus scanners. From this, we can say that our proposed framework is more efficient in detecting malware rather than the manually created signatures of many anti-virus scanners. In addition to that, we also compare the complexity analysis of our proposed framework i.e., FSDroid with distinct anti-virus scanners and experiment were performed on 1000 distinct Android apps that were collected from real-world and empirical result reveals that our proposed framework is able to detect malware-infected apps in less time period when compared to distinct anti-virus scanners.

9.5.3 Detection of known and unknown malware families

Detection of known malware families In this section, we check that our proposed framework is capable to detect malware of known family or not. For this experiment, we select 20 sample of each families (in our study, we consider sample of 81 different families shown in Table 20.) and train it with our selected model. LSSVM with RBF kernel is capable to detect an average of 98.7% malware apps. The name of families and the samples used for

³¹To perform experiment we collect 1000 distinct Android apps from real-world

Table 17 Mean difference between performance of different supervised machine learning technique

Accuracy	Accuracy									
	Decision tree analysis	Logistic regression	SVM with linear kernel	SVM with polynomial kernel	SVM with RBF kernel	Neural network	Naïve Bayes classifier	LSSVM with linear kernel	LSSVM with polynomial kernel	LSSVM with RBF kernel
Decision tree analysis	0	6.77	6.41	9.81	5.87	5.99	10.21	2.10	-1.11	-2.88
Logistic regression	-6.88	0	-0.88	-4.88	2.88	-1.89	3.88	-4.99	-8.11	-9.88
SVM with linear kernel	-6.55	0.77	0	-4.81	3.87	-0.89	3.21	-4.10	-7.8	-9.41
SVM with polynomial kernel	-2.88	4.71	4.41	0	7.27	3.29	8.01	-0.19	-3.41	-5.28
SVM with RBF kernel	-9.8	-2.77	-3.41	-7.81	0	-4.89	0.29	-7.10	-11.01	-12.78
Neural network	-5.88	1.77	0.41	-4.81	4.17	0	4.21	-3.10	-7.11	-8.88
Naïve Bayes classifier	-10.78	-3.77	-3.41	-8.10	-0.87	-4.99	0	-8.10	-11.1	-12.80
LSSVM with linear kernel	-2.10	4.97	4.10	0.81	7.87	3.99	8.21	0	-3.91	-5.02
LSSVM with polynomial kernel	1.99	8.17	7.17	3.81	11.71	7.09	12.1	3.03	0	-1.88
LSSVM with RBF kernel	2.88	9.77	9.41	5.81	12.87	8.99	13.21	4.90	1.99	0

Table 18 Comparison with previously developed frameworks/approaches

Framework/ Approach	Goal	Methodology	Deployment	Data Set	Detection rate	Availability
Andromaly [76] (2012)	Detection	Dynamic and Profile-based	Distributed	Very-Limited	High	Free
AndroSimilar [27](2013)	Detection	Static	Off-device	Limited	Moderate	—
Andrubis [51] (2014)	Analysis and Detection	Static, Dynamic, Profile-based and Behavioural	Off-device	Higher	Moderate	Free
Aurasium [89](2012)	Detection	Dynamic and Behavioural	Off-device	Limited	High	Free
CopperDroid [79](2015)	Analysis and Detection	Dynamic, System/API and VMI	Off-Device	Limited	Moderate	Free
Crowdroid [14] (2011)	Detection	Dynamic, System call/API and Behavioural	Distributed	Very-Limited	High	—
Paranoid Android [70] (2010)	Detection	Dynamic and Behavioural	Off-device	Limited	—	—
TaintDroid [26] (2014)	Detection	Dynamic, System call/API and Behavioural	Off-Device	Very-Limited	Moderate	Free
HinDroid [39](2017)	Detection	Dynamic and API	Off-device	Limited	Moderate	—
MalDozer [44](2018)	Detection	Dynamic	Off-Device	Limited	Moderate	—
HEMD[99](2018)	Detection	Dynamic and Permissions	Off-device	Limited	Moderate	—
DroidDet[100](2018)	Detection	Static	Off-device	Limited	Moderate	—
Wei Wang[86](2019)	Detection	Dynamic	Off-device	Limited	Moderate	—
MalInsight[37](2019)	Detection	Dynamic	Off-device	Limited	High	—
FSDroid (our proposed framework)	Detection	Dynamic,Permissions, API calls, user-rating and Number of user download app	Off-device	Unlimited	Higher	Free

Detection rate of our proposed malware detection model (i.e., FSDroid) is higher when compared to distinct frameworks/approaches available in the literature. Frameworks/approaches proposed in the literature developed and tested with limited data set. Experiment was performed with our collected data set and empirical result reveals that our proposed framework has achieved 3% higher detection rate when compared to distinct framework available in the literature.

Table 19 Detection rate of FSDroid with different anti-virus scanners

Name of the Anti-virus	Detection rate (in %)	Speed to detect malware in sec
Cyren	82	60
Ikarus	82.68	62
VIPRE	89	40
McAfee	89	30
AVG	90	32
AVware	92.8	30
ESET NOD32	92.9	20
CAT QuickHeal	96.9	32
AegisLab	97.1	30
NANO Antivirus	96.2	20
FSDroid (our proposed framework)	98.8	12

Detection speed calculated on Android apps whose size is less or equivalent to 50MB. For this experiment, to compare the performance of FSDroid we consider freely available anti-virus in the market. Speed is measured for a particular Android app taken from real-world. Testing was performed with 1000 distinct Android apps collected from promised repositories

Table 20 Top malware families used in our data set

ID	Family	# of samples	ID	Family	# of samples	ID	Family	# of samples
A1	Airpush	500	A2	AndroRAT	140	A3	Andup	300
A4	Aples	120	A5	BankBot	100	A6	Bankun	133
A7	Boqx	130	A8	Boxer	122	A9	Cova	100
A10	Dowgin	100	A11	DroidKungFu	100	A12	Erop	120
A13	FakeAngry	110	A14	FakeAV	120	A15	FakeDoc	120
A16	FakeInst	110	A17	FakePlayer	120	A18	FakeTimer	120
A19	FakeUpdates	120	A20	Finspy	1110	A21	Fjcon	1230
A22	Fobus	1020	A23	Fusob	1810	A24	GingerMaster	1920
A25	GoldDream	200	A26	Gorpo	120	A27	Gumen	200
A28	Jisut	620	A29	Kemoge	720	A30	Koler	200
A31	Ksapp	290	A32	Kuguo	100	A33	Kyview	500
A34	Leech	30	A35	Lnk	100	A36	Lotoor	20
A37	Mecor	29	A38	Minimob	33	A39	Mmarketpay	200
A40	MobileTX	50	A41	Mseg	23	A42	Mtk	20
A43	Nandrobox	10	A44	Obad	100	A45	Opfake	120
A46	Penetho	120	A47	Ramnit	120	A48	Roop	120
A49	RuMMS	100	A50	SimpleLocker	110	A51	SlemBunk	120
A52	SmsKey	120	A53	SMsZombie	110	A54	Spambot	115
A55	SpyBubble	120	A56	Stealer	300	A57	Steek	230
A58	Svpeng	20	A59	Tesbo	21	A60	Triada	200
A61	Univert	210	A62	UpdtKiller	100	A63	Utchi	300
A64	Vidro	92	A65	VikingHorde	230	A66	Vmvol	533
A67	Winge	190	A68	Youmi	689	A69	Zitmo	230
A70	Ztorg	1000	A71	Imlog	50	A72	SMSreg	50
A73	Gappusin	50	A74	Adrd	50	A75	Geinimi	100
A76	Kmin	157	A77	Plankton	125	A78	GingerMaster	100
A79	Iconosys	100	A80	SendPay	18	A81	GoldDream	200

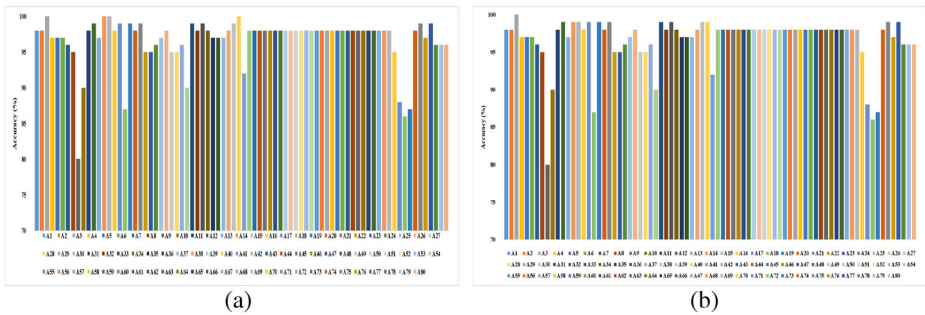


Fig. 15 Detection rate of proposed framework LSSVM with RBF kernel (i.e., FSDroid)

each family can be found in Table 20 and the detection rate of our proposed framework for each family is illustrated in Figs. 15a and b.

Detection of unknown malware families To check whether the LSSVM with RBF kernel is capable to detect unknown malware families or not, we trained, our proposed framework with the random selection of 10 different families obtained by principle of counting and test is applied on the rest of the remaining 71 families present in the data set. Table 21 shows the result of LSSVM with RBF kernel when we train with 10 selected families. From Table 21, we can say that if we train LSSVM with RBF kernel function and having few number of known families samples which are necessary to generalize the behavior of most malware families it gives better detection rate.

Table 21 Detection of FSDroid to detect Unknown malware families

Combination of Android malware families to trained the model	Detection rate when trained LSSVM with RBF kernel
{A1,A2,A3,A4,A5,A6,A7,A8,A9,A10}	66%
{A1,A3,A4,A5,A6,A7,A8,A8,A10,A11}	70%
:	:
:	:
{A2,A3,A4,A5,A6,A7,A8,A9,A10,A11}	59%
:	:
:	:
:	:
:	:
:	:
{A7,A13,A42,A55,A67,A37,A68,A79,A22,A51}	98.4%
:	:
:	:
:	:
:	:
:	:

In summary, our proposed framework is capable to detect Android malware more effectively when compared with several anti-virus scanners which regularly update their signature definition. In addition, our proposed framework is capable to identify Android malware more efficiently whenever we trained with limited number of malware families.

9.5.4 Experimental findings

The comprehensive conclusion of our experimental work is presented in this section. The empirical study was conducted for thirty different categories of Android apps by considering LSSVM with three distinct kinds of kernel functions i.e., linear, RBF and polynomial. On the basis of the experimental results, this research paper is able to answer the questions mentioned in Section 2.

- RQ1:** In this paper, we applied three distinct machine learning algorithms to build a model to detect whether the app is benign or malware. On the basis of Tables 8-13, it can be implicit that model build by employing LSSVM with RBF kernel by using selected set of features obtained as a result of FS4 as an input gives better outcome when compared to others.
- RQ2:** To respond the RQ2, Fig. 15 and Tables 18 and 19 were analyzed. Here, it is found that the model build by utilizing LSSVM with RBF kernel is capable to detect malware from real-world apps.
- RQ3:** In the present paper, four distinct kind of feature subset selection approaches and six distinct kind of feature ranking approaches are used to identify the smaller subset of features. By utilizing these approaches, we considered best possible subsets of the features which helps to build a model to identify that either the app is benign or malware. On the basis of the Tables 8-13, in number of cases there occurs a reduced subset of features which are best for building a detection model when compared to all the extracted features.
- RQ4:** In the present paper, six distinct variants of feature ranking approaches are used to discover the reduced subset of features. On the basis of *t*-test study, it is seen that feature selection by implementing PCA i.e., FR6 approach gives the better outcomes when matched to others approaches.
- RQ5:** For this paper, four distinct kind of feature subset selection approaches are used to find the reduced subset of features. On the basis of *t*-test study, it is seen that feature selection by utilizing FS4 gives the outcomes which are persuasively better when compared to other approaches.
- RQ7:** For this work, pair-wise *t*-test being utilized to identify whether feature subset selection approaches perform better than feature ranking approaches or both of them carried out equally well. On the basis of *t*-test outcomes it is seen that, there is a relevance difference among feature subset selection and feature ranking approach. Moreover, the value of mean difference shows that feature ranking gives better results than the feature subset selection approaches.
- RQ7:** On the basis of Section 9, we can observe that the performance of the feature selection approaches vary by using the distinct machine learning techniques. Further, it also observed that selection of machine learning algorithm to build a malware detection model which detect either the app is malware or not is based on the feature selection approaches.

10 Threat to validity

In this section, threats to validity which are experienced at the time of performing the experiment are presented. Below we discuss them:

- i. *Construct validity* : In this work, presented models for malware detection only detect either an app is benign or malware, but does not state that how many number of possible permissions and API calls are required to detect malware.
- ii. *External Validity*: Cyber-criminals develops malware on daily basis to misuse the user information. In this work, we considered 81 different malware families to train the model and our proposed model is capable to detect malware from known and unknown families. Further, research can be extended to train model with more malware families and which is capable to detect more malware apps from real-world.
- iii. *Internal Validity* : The third threat lies in the consistency of the data used in this study. We collected data from different promised repositories mentioned in Section 2.3. Any error in the information not mentioned in the sources were not considered in this work. Also we can not claim that the data considered for experiment is 100% accurate, we believed that it has been collected consistently.

11 Conclusion

This work emphasizes on designing a malware detection framework by using selected set of features which help us to identify that an Android app belong to malware class or benign class. The execution process was performed by taking assistance of thirty different categories of Android apps.

Our submissions after performing the experiment are following:

- Empirical results indicate that, it is feasible to determine a small subset of features. The malware detection model build by considering this determined set of features is able to detect malware and benign apps with inferior value of misclassified errors and better accuracy.
- On the basis of experimental finding, we observed that after considering feature selection approaches it helps to reduced the feature sets. The result of models build by using feature selection approaches perform better when compared to all extracted feature sets.
- The AA, BU, LS, PE, RA, TO set of features were found to be relevance detectors for malware detection by utilizing feature selection approaches.
- On the basis of *t*-test study, it is seen that there has been no relevant difference among implemented feature selection approaches. Moreover, the models build by utilizing distinct machine learning techniques are relevantly distinct.
- On the basis of mean difference, it is seen that model build by considering selected set of features as an input gives better detection rate when compared to model build by considering all set of extracted features. Moreover, the model build by utilizing LSSVM with RBF kernel gives better outcomes when compared to other techniques.
- On the basis of *Cost-benefit* analysis, we implicit that the selected features by utilizing FS4, achieved high median *Cost-Benefit* value when compared to other approaches.
- The results of malware detection model, is influenced by the feature selection approaches.

- On the basis of proposed detection framework, it is seen that model build by utilizing LSSVM with RBF kernel is capable to detect 98.75% unknown malware from real-world apps.
- Proposed framework is able to detect 98.8% malware-infected Android apps when compared to commercial anti-virus softwares and in addition to that it also achieved 3% higher detection rate when compared to different frameworks or approaches proposed in the literature.

In this work, proposed models for malware detection only detect that either an app is malware or benign. Further, work can be extended to develop a model for malware detection which predict whether a particular feature is capable to detect malware or not. Moreover, this study can be replicated over other Android apps repository which utilized soft computing models to attain better detection rate for malware.

References

1. Aafer Y, Du W, Yin H (2013) Droidapiminer: Mining api-level features for robust malware detection in android. In: International conference on security and privacy in communication systems, Springer, pp 86–103
2. Allix K, Bissyandé TF, Jérôme Q, Klein J, Traon YL et al (2016) Empirical assessment of machine learning-based malware detectors for android. *Empir Softw Eng* 21(1):183–211
3. Arora A, Peddoju SK, Conti M (2019) Permpair: Android malware detection using permission pairs. *IEEE Trans Inform Forensics Secur* 15:1968–1982
4. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens C (2014) Drebin: Effective and explainable detection of android malware in your pocket. In: *Ndss*, vol 14, pp 23–26
5. Aubery-Derrick S (2011) Detection of smart phone malware. Unpublished PhD Thesis Electronic and Information Technology University Berlin, pp 1–211
6. Azmoodeh A, Dehghantanha A, Choo KKR (2018) Robust malware detection for internet of (battle-field) things devices using deep eigenspace learning. *IEEE Trans Sustain Comput* 4(1):88–95
7. Backes M, Gerling S, Hammer C, Maffei M, von Styp-Rekowsky P (2013) Appguard—enforcing user requirements on android apps. In: International conference on TOOLS and Algorithms for the construction and analysis of systems. Springer, pp 543–548
8. Barrera D, Kayacik HG, Oorschot PCV, Somayaji A (2010) A methodology for empirical analysis of permission-based security models and its application to android. In: Proceedings of the 17th ACM conference on Computer and communications security, pp 73–84
9. Bhandari S, Gupta R, Laxmi V, Gaur MS, Zemmari A, Anikeev M (2015) Draco: Droid analyst combo an android malware analysis framework. In: Proceedings of the 8th international conference on security of information and networks, pp 283–289
10. Bhattacharya A, Goswami RT (2018) A hybrid community based rough set feature selection technique in android malware detection, Springer
11. Birendra C (2016) Android permission model. arXiv:160704256
12. Bläsing T, Batyuk L, Schmidt AD, Camtepe SA, Albayrak S (2010) An android application sandbox system for suspicious software detection. In: 2010 5th International conference on malicious and unwanted software, IEEE, pp 55–62
13. Bugiel S, Davi L, Dmitrienko A, Fischer T, Sadeghi AR, Shastri B (2012) Towards taming privilege-escalation attacks on android. In: *NDSS*, vol 17, p 19
14. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp 15–26
15. Cai H, Meng N, Ryder B, Yao D (2018) Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Trans Inf Forensics Secur* 14(6):1455–1470
16. Chaikla N, Qi Y (1999) Genetic algorithms in feature selection. In: IEEE SMC'99 conference proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028), IEEE, vol 5, pp 538–540

17. Chakradeo S, Reaves B, Traynor P, Enck W (2013) Mast: Triage for market-scale mobile malware analysis. In: Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks, pp 13–24
18. Chen KZ, Johnson NM, D’Silva V, Dai S, MacNamara K, Magrino TR, Wu EX, Rinard M, Song DX (2013) Contextual policy enforcement in android applications with permission event graphs. In: NDSS, p 234
19. Chen S, Xue M, Fan L, Hao S, Xu L, Zhu H, Li B (2018) Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *Comput Secur* 73:326–344
20. Chen Y, Xiong J, Xu W, Zuo J (2019) A novel online incremental and decremental learning algorithm based on variable support vector machine. *Clust Comput* 22(3):7435–7445
21. Cruz AEC, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE, pp 460–463
22. Desnos A et al (2013) Androguard-reverse engineering, malware and goodware analysis of android applications. URL code google.com/p/androguard 153
23. DeviPriya K, Lingamgunta S (2020) Multi factor two-way hash-based authentication in cloud computing. *Int J Cloud Appl Comput (IJCAC)* 10(2):56–76
24. Dini G, Martinelli F, Saracino A, Sgandurra D (2012) Madam: a multi-level anomaly detector for android malware. In: International conference on mathematical methods, models, and architectures for computer network security. Springer, pp 240–253
25. Enck W, Ongtang M, McDaniel P (2009) On lightweight mobile phone application certification. In: Proceedings of the 16th ACM conference on Computer and communications security, pp 235–245
26. Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, McDaniel P, Sheth AN (2014) Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans Comput Syst (TOCS)* 32(2):1–29
27. Faruki P, Ganmoor V, Laxmi V, Gaur MS, Bharmal A (2013) Androsimilar: robust statistical feature signature for android malware detection. In: Proceedings of the 6th International conference on security of information and networks, pp 152–159
28. Faruki P, Bharmal A, Laxmi V, Ganmoor V, Gaur MS, Conti M, Rajarajan M (2014) Android security: a survey of issues, malware penetration, and defenses. *IEEE Commun Surv Tutor* 17(2):998–1022
29. Felt AP, Ha E, Egelman S, Haney A, Chin E, Wagner D (2012) Android permissions: User attention, comprehension, and behavior. In: Proceedings of the eighth symposium on usable privacy and security, pp 1–14
30. Fereidooni H, Conti M, Yao D, Sperduti A (2016) Anastasia: Android malware detection using static analysis of applications. In: 2016 8th IFIP international conference on new technologies, mobility and security (NTMS). IEEE, pp 1–5
31. Fuchs AP, Chaudhuri A, Foster JS (2009) Scandroid: Automated security certification of android applications. Manuscript, Univ of Maryland, <http://www.cs.umd.edu/avik/projects/scandroidasca> 2(3)
32. Gadekallu TR, Rajput VS, Reddy MPK, Lakshmana K, Bhattacharya S, Singh S, Jolfaei A, Alazab M (2020) A novel pca-whale optimization-based deep neural network model for classification of tomato plant diseases using gpu. *J. Real-Time Image Proc.*, 1–14
33. Gao K, Khoshgoftar TM, Napolitano A (2009) Exploring software quality classification with a wrapper-based feature ranking technique. In: 2009 21st IEEE international conference on tools with artificial intelligence, IEEE, pp 67–74
34. Grace M, Zhou Y, Zhang Q, Zou S, Jiang X (2012a) Riskranker: scalable and accurate zero-day android malware detection. In: Proceedings of the 10th international conference on Mobile systems, applications, and services, pp 281–294
35. Grace MC, Zhou Y, Wang Z, Jiang X (2012b) Systematic detection of capability leaks in stock android smartphones. In: NDSS, vol 14, p 19
36. Gupta BB, Perez GM, Agrawal DP, Gupta D (2020) Handbook of computer networks and cyber security. Springer, Berlin
37. Han W, Xue J, Wang Y, Liu Z, Kong Z (2019) Malinsight: A systematic profiling based malware detection framework. *J Netw Comput Appl* 125:236–250
38. He S, Li Z, Tang Y, Liao Z, Li F, Lim SJ (2020) Parameters compressing in deep learning. *Comput Mater Contin* 62(1):321–336
39. Hou S, Ye Y, Song Y, Abdulhayoglu M (2017) Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International conference on knowledge discovery and data mining, pp 1507–1515

40. Jeon J, Micinski KK, Vaughan JA, Fogel A, Reddy N, Foster JS, Millstein T (2012) Dr. android and mr. hide: fine-grained permissions in android applications. In: Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices, pp 3–14
41. Jerlin MA, Marimuthu K (2018) A new malware detection system using machine learning techniques for api call sequences. *Journal of Applied Security Research* 13(1):45–62
42. Jiang S, Chen W, Li Z, Yu H (2019) Short-term demand prediction method for online car-hailing services based on a least squares support vector machine. *IEEE Access* 7:11882–11891
43. Kadir AFA, Stakhanova N, Ghorbani AA (2015) Android botnets: What urls are telling us. In: International conference on network and system security, Springer, pp 78–91
44. Karbab EB, Debbabi M, Derhab A, Mouheb D (2018) Maldozer: Automatic framework for android malware detection using deep learning. *Digit Investig* 24:S48–S59
45. Khare N, Devan P, Chowdhary CL, Bhattacharya S, Singh G, Singh S, Yoon B (2020) Smo-dnn: Spider monkey optimization and deep neural network hybrid classifier model for intrusion detection. *Electronics* 9(4):692
46. Kirubavathi G, Anitha R (2018) Structural analysis and detection of android botnets using machine learning techniques. *Int J Inf Secur* 17(2):153–167
47. Kohavi R, John GH et al (1997) Wrappers for feature subset selection. *Artificial intelligence* 97(1-2):273–324
48. Kumar L, Sripada SK, Sureka A, Rath SK (2018) Effective fault prediction model developed using least square support vector machine (lssvm). *J Syst Softw* 137:686–712
49. Letteri I, Penna GD, Gasperis GD (2019) Security in the internet of things: botnet detection in software-defined networks by deep learning techniques. *Int J High Perform Comput Netw* 15(3-4):170–182
50. Li L, Bissyandé TF, Papadakis M, Rasthofer S, Bartel A, Oceau D, Klein J, Traon L (2017) Static analysis of android apps: A systematic literature review. *Inf Softw Technol* 88:67–95
51. Lindorfer M, Neugschwandtner M, Weichselbaum L, Fratantonio Y, Veen VVD, Platzer C (2014) Andrubis–1,000,000 apps later: A view on current android malware behaviors. In: 2014 third international workshop on building analysis datasets and gathering experience returns for security (BADGERS). IEEE, pp 3–17
52. Loorak MH, Fong PW, Carpendale S (2014) Papilio: Visualizing android application permissions. In: Computer graphics forum, Wiley Online Library, vol 33, pp 391–400
53. Ma Z, Ge H, Liu Y, Zhao M, Ma J (2019) A combination method for android malware detection based on control flow graphs and machine learning algorithms. *IEEE Access* 7:21235–21245
54. Mahindru A, Sangal A (2019) Deepdroid: Feature selection approach to detect android malware using deep learning, IEEE
55. Mahindru A, Sangal A (2020a) Dldroid: Feature selection based malware detection framework for android apps developed during covid-19. *Int J Emerg Technol*
56. Mahindru A, Sangal A (2020b) Feature-based semi-supervised learning to detect malware from android. In: Automated software engineering: a deep learning-based approach, Springer, pp 93–118
57. Mahindru A, Sangal A (2020c) Gadroid: A framework for malware detection from android by using genetic algorithm as feature selection approach. *Int J Adv Sci Technol* 29(5):5532–5543
58. Mahindru A, Sangal A (2020d) Parudroid: Validation of android malware detection dataset. *J Cybersecur Inform Manag* 3(2):42–52
59. Mahindru A, Sangal A (2020e) Perbdroid: Effective malware detection model developed using machine learning classification techniques. In: A journey towards bio-inspired techniques in software engineering, Springer, pp 103–139
60. Mahindru A, Singh P (2017) Dynamic permissions based android malware detection using machine learning techniques. In: Proceedings of the 10th innovations in software engineering conference, pp 202–210
61. Matsudo T, Kodama E, Wang J, Takata T (2012) A proposal of security advisory system at the time of the installation of applications on android os. In: 2012 15th International conference on network-based information systems, IEEE, pp 261–267
62. Narayanan A, Chandramohan M, Chen L, Liu Y (2018) A multi-view context-aware approach to android malware detection and malicious code localization. *Empir Softw Eng* 23(3):1222–1274
63. Narudin FA, Feizollah A, Anuar NB, Gani A (2016) Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput* 20(1):343–357
64. Novakovic J (2010) The impact of feature selection on the accuracy of naïve bayes classifier. In: 18th Telecommunications forum TELFOR, vol 2, pp 1113–1116
65. Ongtang M, McLaughlin S, Enck W, McDaniel P (2012) Semantically rich application-centric security in android. *Secur Commun Netw* 5(6):658–673
66. Pawlak Z (1982) Rough sets. *Int J Comput Inform Sci* 11(5):341–356

67. Peiravian N, Zhu X (2013) Machine learning for android malware detection using permission and api calls. In: 2013 IEEE 25th international conference on tools with artificial intelligence. IEEE, pp 300–305
68. Petsas T, Voyatzis G, Athanasopoulos E, Polychronakis M, Ioannidis S (2014) Rage against the virtual machine: hindering dynamic analysis of android malware. In: Proceedings of the seventh european workshop on system security, pp 1–6
69. Plackett RL (1983) Karl pearson and the chi-squared test. In: International statistical review/Revue Internationale de Statistique, pp 59–72
70. Portokalidis G, Homburg P, Anagnostakis K, Bos H (2010) Paranoid android: versatile protection for smartphones. In: Proceedings of the 26th annual computer security applications conference, pp 347–356
71. Rastogi V, Chen Y, Enck W (2013) Appsplayground: automatic security analysis of smartphone applications. In: Proceedings of the third ACM conference on Data and application security and privacy, pp 209–220
72. Razak MFA, Anuar NB, Othman F, Firdaus A, Afifi F, Salleh R (2018) Bio-inspired for features optimization and malware detection. Arab J Sci Eng 43(12):6963–6979
73. Rosen S, Qian Z, Mao ZM (2013) Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users. In: Proceedings of the third ACM conference on Data and application security and privacy, pp 221–232
74. Sanz B, Santos I, Laorden C, Ugarte-Pedrero X, Bringas PG, Álvarez G (2013) Puma: Permission usage to detect malware in android. In: International joint conference CISIS' 12-ICEUTE 12-SOCO 12, Special Sessions, Springer, pp 289–298
75. Saracino A, Scandurra D, Dini G, Martinelli F (2016) Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Trans Dependable Secure Comput 15(1):83–97
76. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y (2012) “andromaly”: a behavioral malware detection framework for android devices, vol 38, pp 161–190
77. Shahzad F, Akbar M, Khan S, Farooq M (2013) Tstructdroid: Realtime malware detection using in-execution dynamic analysis of kernel process control blocks on android. National University of Computer & Emerging Sciences, Islamabad, Pakistan, Tech Rep
78. Suykens JA, Brabanter JD, Lukas L, Vandewalle J (2002) Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing 48(1-4):85–105
79. Tam K, Khan SJ, Fattori A, Cavallaro L (2015) Copperdroid: Automatic reconstruction of android malware behaviors. In: Ndss
80. Tan Y, Xue Y, Liang C, Zheng J, Zhang Q, Zheng J, Li Y (2018) A root privilege management scheme with revocable authorization for android devices. J Netw Comput Appl 107:69–82
81. Mas'ud MZ, Sahib S, Abdollah MF, Selamat SR, Yusof R (2014) Analysis of features selection and machine learning classifier in android malware detection, IEEE
82. Wang C, Xu Q, Lin X, Liu S (2019a) Research on data mining of permissions mode for android malware detection. Clust Comput 22(6):13337–13350
83. Wang D, Romagnoli J (2005) Robust multi-scale principal components analysis with applications to process monitoring. J Process Control 15(8):869–882
84. Wang W, Wang X, Feng D, Liu J, Han Z, Zhang X (2014) Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans Inf Forensics Secur 9(11):1869–1882
85. Wang W, Li Y, Wang X, Liu J, Zhang X (2018) Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. Future Gener Comput Syst 78:987–994
86. Wang W, Zhao M, Wang J (2019b) Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J Ambient Intell Humaniz Comput 10(8):3035–3043
87. Wu DJ, Mao CH, Wei TE, Lee HM, Wu KP (2012) Droidmat: Android malware detection through manifest and api calls tracing. In: 2012 Seventh Asia joint conference on information security, IEEE, pp 62–69
88. Xiao X, Zhang S, Mercaldo F, Hu G, Sangaiah AK (2019) Android malware detection based on system call sequences and lstm. Multimed Tools Appl 78(4):3979–3999
89. Xu R, Saïdi H, Anderson R (2012) Aurasium: Practical policy enforcement for android applications. In: Presented as part of the 21st {USENIX} security symposium ({USENIX} security, vol 12, pp 539–552
90. Yamaguchi S, Gupta B (2020) Malware threat in internet of things and its mitigation analysis. In: Security, privacy, and forensics issues in big data. IGI Global, pp 363–379
91. Yan LK, Yin H (2012) Droidscape: Seamlessly reconstructing the {OS} and dalvik semantic views for dynamic android malware analysis. In: Presented as part of the 21st {USENIX} security symposium ({USENIX} security, vol 12, pp 56–584

92. Yerima SY, Sezer S, McWilliams G, Muttik I (2013) A new android malware detection approach using bayesian classification, IEEE, AINA
93. Yerima SY, Sezer S, McWilliams G (2014) Analysis of bayesian classification-based approaches for android malware detection. *IET Inf Secur* 8(1):25–36
94. Zhang LB, Peng F, Qin L, Long M (2018) Face spoofing detection based on color texture markov feature and support vector machine recursive feature elimination. *J Vis Commun Image Represent* 51:56–69
95. Zheng C, Zhu S, Dai S, Gu G, Gong X, Han X, Zou W (2012) Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. In: *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices*, pp 93–104
96. Zhou S, Tan B (2020) Electrocardiogram soft computing using hybrid deep learning cnn-elm. *Appl Soft Comput* 86:105778
97. Zhou W, Zhou Y, Jiang X, Ning P (2012) Detecting repackaged smartphone applications in third-party android marketplaces. In: *Proceedings of the second ACM conference on data and application security and privacy*, pp 317–326
98. Zhou Y, Jiang X (2012) Dissecting android malware: Characterization and evolution. In: *2012 IEEE symposium on security and privacy, IEEE*, pp 95–109
99. Zhu HJ, Jiang TH, Ma B, You ZH, Shi WL, Cheng L (2018a) Hemd: a highly efficient random forest-based malware detection framework for android. *Neural Comput and Applic* 30(11):3353–3361
100. Zhu HJ, You ZH, Zhu ZX, Shi WL, Chen X, Cheng L (2018b) Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* 272:638–646

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.