Hindawi
Security and Communication Networks
Volume 2022, Article ID 5339926, 24 pages
https://doi.org/10.1155/2022/5339926

Research Article

BFEDroid: A Feature Selection Technique to Detect Malware in Android Apps Using Machine Learning

Collins Chimeleze,[1] **Norziana Jamil**,[1] **Roslan Ismail**,[1] **Kwok-Yan Lam**,[2] **Je Sen Teh**,[3] **Joshua Samual**,[4] **and Chidiebere Akachukwu Okeke**[5]

[1]*College of Computing and Informatics, Universiti Tenaga Nasional, Selangor, Malaysia*
[2]*School of Computer Science and Engineering, Nanyang Technological University, Singapore*
[3]*School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Pulau Pinang, Malaysia*
[4]*School of Technology, Faculty of Computing, Engineering & Technology, Asia Pacific University of Technology and Innovation, Kuala Lumpur, Malaysia*
[5]*Department of Electrical and Electronics Engineering, Faculty of Engineering, Universiti Putra Malaysia, Seri Kembangan, Malaysia*

Correspondence should be addressed to Collins Chimeleze; chimeleze@uniten.edu.my

Received 17 June 2022; Revised 18 September 2022; Accepted 27 September 2022; Published 11 October 2022

Academic Editor: Mian Ahmad Jan

Malware detection refers to the process of detecting the presence of malware on a host system, or that of determining whether a specific program is malicious or benign. Machine learning-based solutions first gather information from applications and then use machine learning algorithms to develop a classifier that can distinguish between malicious and benign applications. Researchers and practitioners have long paid close attention to the issue. Most previous work has addressed the differences in feature importance or the computation of feature weights, which is unrelated to the classification model used, and therefore, the implementation of a selection approach with limited feature hiccups, and increases the execution time and memory usage. BFEDroid is a machine learning detection strategy that combines backward, forward, and exhaustive subset selection. This proposed malware detection technique can be updated by retraining new applications with true labels. It has higher accuracy (99%), lower memory consumption (1680), and a shorter execution time (1.264SI) than current malware detection methods that use feature selection.

1. Introduction

The feature selection process is an important process in high-dimensional data mining applications. It involves selecting a subset of relevant features and applying them to the given learning algorithm. The benefits of using feature selection methods include data reduction and better visualization of the trend of the data. It can also be used to handle noisy and irrelevant data in the dataset to obtain accurate results.

Methods of feature selection are categorized according to their organization of search: exponential, sequential, and random. (i) Generation of successors (subset) is as follows: there are five possible operations for generating successors: forward, backward, compound, weighted, and random selection. (ii) Evaluation measure is as follows: theseinclude the probability of error, divergence, dependence, interclass distance, uncertainty of the information, and consistency evaluation.

Subset search algorithms iterate over possible feature subsets, as guided by a unique evaluation measure that captures the quality of each subset. The number of reduced features and their influence on learning can be evaluated, analyzed, and compared by using many methods. To define the goal of learning concepts of the process, the feature selection technique that is used should choose the best feature subset from the feature space. The following variables must be considered during feature selection: (1) starting point, (2) search strategy, (3) evaluation of subsets, and (4) stopping criteria. A comparison of feature selection

algorithms based on these parameters is shown in Table1. We define techniques of feature selection to provide a comparative analysis of the relevant methods in terms of search organization, feature generation, and measure of evaluation. This can help practitioners choose a technique that is appropriate for their goals and resources.

Section 2 provides a brief survey of literature in the area. Section 3 provides a detailed description of the proposed work (BFEDroid), and how the models are designed. Section 4 reports the accuracy of the proposed method. Section 5 summarizes the findings of this study and conclusions, and offers directions for future work in the area.

2. Literature Survey

Mahindru and Sangal [1] stressed the need for building a framework for malware detection by utilizing a limited collection of criteria that can help us determine whether an Android app contains malware or is benign. The framework was executed by 30 kinds of Android applications. A model subsequently developed by using the nonlinear ensemble decision tree, forest approach, multilayer perception, deep neural network, and farthest first clustering was able to identify 98.8% of malware in the given apps. The work in this study can be improved by constructing a model for malware detection that predicts whether a certain attribute is capable of identifying malware. Furthermore, Mahindru and Sangal can be duplicated across Android app repositories that use soft computing models to obtain a higher rate of detection of malware. Ma et al. [2] reported a method for detecting Android malware, which decompiled Android applications and built a CFG of each. They then developed three system API datasets based on the CFG: datasets of API usage (the API contained in the CFG), API frequency (number of times the CFG uses the corresponding API), and API sequence (the API sequence appearing in the CFG). They then built a dataset of API sequences for Android applications and created a two-class classification model for each data collection to determine whether the incoming application was malicious. Next, the authors evaluated the accuracy of each model by using standard classification metrics, that is, precision, recall, and F-score, to compare the performance of the three models. Finally, they used a combination of the models. Finally, they used a combination of methods to create an ensemble model that achieved a precision of detection of malicious apps of 98.98%.

Cai et al. [3] developed a feature weighting technique (JOWM) and a system to detect malware in Android apps (JOWMDroid). The JOWMDroid scheme uses static analysis in five steps. APK files are first used to extract the original features in eight categories. IG is then used to choose a subset of features with the most important characteristics. Following this, three ML models are used to generate an initial weight for each selected characteristic. Five weight mapping functions have been developed to map the beginning weight of each feature to its end weight. Finally, the DE approach is used to jointly optimize the weight mapping function and parameters of the classifier. Cat et al. also compared the performance of four weight-aware classifiers and four cutting-edge feature weighting techniques with that of the JOWM. Their experimental findings showed that the fundamental weights performed very well. Furthermore, changes in weight mapping and the joint parameter significantly enhanced the accuracy of malware detection. Their approach outperformed the four cutting-edge methods against which it was compared. Furthermore, a combined optimization could quickly yield a set of optimum settings. Weight-aware classifiers outperform weight-unaware classifiers when the appropriate feature weighting approach is used.

Fallah and Bidgoly [4] investigated machine learning methods in various ways: the number of characteristics that needed to be learned, the kind of machine learning techniques used, the volume of recorded data, the ability to identify the family of malware, and the ability to recognize a new type of malware family. The sensitivity of the assessed techniques, namely, decision tree, random forest, KNN, linear regression, SVM, MLP, and Gaussian naive Bayes, to the number of attributes was examined. The algorithms performed poorly in terms of identifying the malware families even when ideal conditions from previous evaluations were used. The machine learning algorithms considered thus had a limited ability to learn. Alzaylaee et al. [5] introduced DL-Droid, an automated framework for the dynamic detection of malware in Android apps. It involves deep learning with a state-based method of input generation, but it can also use the state-of-the-art Monkey tool (stateless method). The authors tested DL-Droid on 31,125 Android applications using 420 static and dynamic characteristics, and compared its performance with that of classical machine learning classifiers and prevalent DL-based frameworks. The results clearly showed that DL-Droid outperformed the prevalent deep learning-based frameworks to detect malware in Android apps in terms of accuracy. To the best of our knowledge, this is the first study to use deep learning to examine the dynamic characteristics of apps. Their findings also highlight the importance of improving input generation for a system of dynamic analysis that employs machine learning to detect malware in Android apps.

Rathore et al. [6] proposed a one-of-a-kind single-policy attack for a white-box scenario in which an adversary has complete information about the detection system. They used a single Q-table strategy to build a reinforcement agent that launches an aggressive attack. The assault had a rate of average fooling of 44.28% with eight models of detection, with a maximum of five changes. When the parameters were equivalent, the attack had the highest fooling rate (54.92%) compared with the DT model, while the GB had the lowest fooling rate (37.77%). Overall, the experimental results demonstrated that even with minimal adjustments, a single-policy assault may successfully evade systems of malware detection to achieve a high fooling rate. The authors also developed a cutting-edge adversarial strategy: a multipolicy assault in grey-box environments in which the attacker does not know the model architecture and the process of classification. This multipolicy approach yielded the highest fooling rate for the DT model (86.09%) with a maximum of five changes, followed by the ET model (75.23%). Even with

Table 1: LSSVM linear kernel (classifier) with rough set analysis (FS4) and principal component analysis (PCA) (FR6) (detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.0793	0.92	**0.721**	0.92	0.90	0.90	0.50	0.090	0.9054	1510
DS2	0.0900	0.91	**0.832**	0.91	0.89	0.89	0.50	0.099	0.8921	1650
DS3	0.0793	0.92	**1.095**	0.92	0.90	0.90	0.50	0.090	0.9073	1824
DS4	0.0400	0.96	**1.450**	0.96	0.94	0.94	0.50	0.056	0.9414	2010
DS5	0.0489	0.95	**1.520**	0.95	0.95	0.95	0.50	0.047	0.9544	2175

limited knowledge, the average fooling rate rises to 53.20%, which is higher than that of the single-policy attack.

Kouliaridis et al. [7] proposed Androtomist, an automated and configurable hybrid analytical tool that combines static analysis with dynamic instrumentation to analyze app behavior on the Android platform. Its outcomes on the three datasets suggest that this dual analysis of mobile applications may significantly improve the detection capabilities of the model. A comparison of static and hybrid analyses resulted in asymmetry, demonstrating that the latter was capable of generating better results in classification across various datasets. Finally, Androtomist provides an easy-to-use environment for almost anyone to analyze mobile apps and is highly configurable, thus allowing researchers to import hooks and scripts for customized dynamic instrumentation.

Wang et al. [8] proposed a technique for detecting malware by analyzing harmful URLs. The vectorization method divides each URL into many segments by using certain characters and then trains the embedding of each segment by using the skip-gram algorithm. This method of vectorization solves the problems of data sparsity and loss of meaning in previous encoding methods. We feed the URL vector into a multiview neural network that can generate numerous views by constructing various values of $S+$ automatically from the input data. Each view focuses on a different part of the data. Combining many views completes the URL-based vector classification. The network favors depth while emphasizing width and can automatically select items from different levels. The authors conducted a series of studies to assess the effectiveness of their strategy and compared it with other methods. Their model performed well on the test set. They also evaluated it in terms of wild malware detection and observed that it was effective.

Manzanares et al. [9] introduced KronoDroid, a data collection of Android malware that combines and supplements data sources, covers a long time span, and describes each sample by using static and dynamic features. Research on and datasets for Android malware have ignored its dynamic nature to present rigid and confined images of it over a short period of time. Notwithstanding, the concept of drift time has received the attention that it deserves in this context. Furthermore, the sources of dynamic data and their characteristics have been overlooked. Abuthawabeh and Mahmoud [10] proposed a model for the detection, classification, and categorization of the family of malware in the Android environment. The model gathers information on conversation-level network traffic from the "CICAndMal2017" dataset, which is both current and empirical. During the feature extraction process, the Peer Shark tool extracts conversion-level features. The dataset is subjected to numerous preprocessing stages. The ensemble learning technique is applied by three feature selection algorithms, namely, random forest, RFE, and light GBM classifiers, to determine the most valuable features. Three classifiers were used to train and assess the built model: the decision tree, random forest, and extra trees.

With insufficient memory and other resource constraints (e.g., CPU availability), many of the state-of-the-art feature subset selection methods cannot be extended to high-dimensional data or datasets with an extremely large volume of instances. In this brief, Ditzler et al. [11] extend online feature selection (OFS), a recently introduced approach that uses partial feature information to make predictions, by developing an ensemble of online linear models. Wang and Shao [12] proposed an improved hybrid feature selection technique (IHFST), which combines a distance evaluation technique (DET), Pearson's correlation analysis, and an ad hoc technique. Lakshmipadmaja and Vishnuvardhan [13] present an improvement to the existing random subset feature selection (RSFS) algorithm for randomly selecting feature subsets and improving stability. Nayar et al. [14] discuss the problems encountered during the feature selection process and how swarm intelligence was used to extract the optimal set of features. Chakraborty and Kawamura [15] proposed a wrapper fitness function that combines classification accuracy with another penalty term that penalizes for a large number of features. Hichem et al. [16] introduced a new binary variant of the grasshopper optimization algorithm and used it for the feature subset selection problem. Saidi et al. [17] present a feature selection method that incorporates the genetic algorithm (GA) and the Pearson correlation coefficient (PCC). Yu et al. [18] examine the effectiveness of feature selection in CPDP using feature subset selection and feature ranking approaches. In contrast to conventional feature selection methods that only focus on finding a single discriminating feature, Mao and Yang [19] present a multilayer feature subset selection method that uses randomized searches and multilayer structures to select discriminative subsets. (MLFSSM) has been proposed. Other influential works include Shukla et al. [20].

Cai et al. [21] introduce DroidCat, a new dynamic app classification technology that complements existing approaches. By using various dynamic functions based on method calls and ICC (intercomponent communication) intents, DroidCat is static and provides greater robustness than dynamic approaches that rely on approaches and system calls. Cai [22] plan to build an infrastructure that can systematically and continuously mine the mobile software

ecosystem. This infrastructure is then used to conduct large-scale longitudinal characterization studies of the ecosystem to understand its evolutionary dynamics. This includes mobile platforms, user apps built on top of the platform, and users (including end users and developers) connecting to the apps. In addition, the characterization results enable proactive app quality and sustainable app security. Kim et al. [23] introduce a reliable malware detection method, including zero-day attacks, and generated and learned fake malware, called transferred deep-convolutional generative adversarial network (tDCGAN), to detect real malware. Sohi et al. [24] propose a new way to distinguish such traffic generation, which has been identified in the literature as one of the main obstacles in evaluating the effectiveness of NIDS. To address these issues, we introduced RNNIDS. RNNIDS applies recurrent neural networks (RNNs) to find complex patterns of attacks and generate similar patterns. By using advanced MLA such as deep learning, you can entirely avoid the feature engineering phase.

Vinayakumar et al. [25] recently published a research study in this direction that demonstrated the algorithm's performance with biased training data, limiting its practical use in real-time situations. As reported in [23], 76% of successful attacks on enterprise endpoints in 2018 were based on zero-day sampling. Anticipating these types of attacks and preparing solutions are an open challenge. This document presents a deep generative adversarial network for generating signatures for invisible malware samples. Moti et al. [26] generated signatures that may resemble malware samples released in the future. Sharmeen et al. [27] propose two different detection models using a semi-supervised approach of deep learning and adaptive frameworks. Wang and Zheng [28] evaluate the performance of different one-class feature selection and classification methods for zero-day Android malware detection. Wen and Chow [29] propose using a CNN-based model to detect malware from very small sequences of binary fragments in PE files. Wu and Kanai [30] propose an Android malware detection method based on deep learning. It uses obfuscation labels in training to let a deep learning model learn to detect obfuscation technology and malware features simultaneously from part of the input. The features with similar behaviors in the data engineering phase based on prior domain knowledge risk becoming ineffective in the face of new threats [31] isolate this human expertise and instead encapsulate the knowledge in deep learning neural networks without prior knowledge of their malicious properties. Experimental results show that the proposed approach by applying AdaBoost ensemble learning to the random forest classifier as the regular classifier achieves 92% F-measure and 95% TPR, and is superior to zero-day malware detection using only the top. Despite using a small number of microarchitectural features captured at run-time by existing HPCs, He et al. [32] propose an ensemble learning-based technique to improve the performance of standard malware detectors. The experimental results show that using only the top four microarchitectural features, our proposed approach of using AdaBoost ensemble learning on the Random Forest classifier as a regular classifier achieves 92% F-measure and 95% TPR with only 2% false positive rate in detecting zero-day malware.

Carlin et al. [33] proposed a new analyzed run-tracking dataset of more than 100,000 labeled samples that would address these shortcomings, and we are making the dataset itself available to the research community to use. Xu et al. [34] provide DroidEvolver, an Android malware detection system that can automatically and continuously update itself when malware is detected without any human intervention. Cai et al. [35] study how benign Android apps execute in every other manner from malware over time, in terms of their execution structure measured via the distribution and interaction among functionality scopes, app components, and callbacks. We systematically characterised the execution structure of malware versus benign apps and revealed previously unknown similarities and disparities by tracing the method calls and inter-component communications (ICCs) of 15,451 benign apps and 15,183 malware developed over eight years (2010-2017). Among other findings, our results show that (1) despite their similarity in execution distribution across functionality scopes, malware accessed framework functionalities primarily through third-party libraries, whereas benign apps were dominated by calls within the framework; (2) use of the activity component has been increasing in malware, while benign apps have seen a continuous drop in such uses; (3) malware invoked significantly more services but significantly fewer content providers than benign apps during both groups' evolution; (4) malware carried ICC data significantly less frequently than benign apps via standard data fields, despite the fact that both groups did not carry any data in most ICCs; and (5) newer malware had a even more distribution of callbacks among event-handler categories, whereas benign apps' distribution remained constant over time. Cai and Ryder [36] propose a longitudinal observation of Android applications to systematically examine how they can be built and executed over time. Through lightweight static assessment and methodical step tracking, we tested the code and execution of 17,664 apps, sampled from advanced apps over the next 8 years, with a high rating of metrics in 3 additional sizes. Our research found that (1) app functionality is heavily dependent on the Android/SDK framework and the dependency continues to grow; (2) persistent add-ons rule utility categories. This is responsible for the maximum number of callbacks in the lifecycle. (3) Callback event handling is increasingly focused on UI occasions rather than utility events. (4) The use of callbacks has generally decreased over time, (5) the majority of user interfaces (ICCs) have now ceased to carry any stat payload, and (6) complex reagents and sink indexes focus on the most effective or dominant events or activities, and ratings of source or sink classes, which have remained relatively stable for the past 8 years.

2.1. Research Gap in Related Works. Previous work in this direction [33] has shown that static analysis cannot unravel the obfuscated code. The previous studies mentioned above have the following limitations: higher memory consumption, limited dataset, higher detection rate on limited dataset, high

computational complexity, implementation of a selection approach, with limited features, limited implementation of 100% classification algorithms on datasets, and no advanced malware detection capabilities. To mitigate these problems, we proposed a new feature-based detection technique. This proposed malware detection technique can be updated by retraining new applications with true labels. We will discuss more about the proposed technique in the methodology section.

2.2. Research Questions. We explore the challenges stated below to develop techniques to select an important subset of features and a model for detecting malware in Android apps with satisfactory accuracy.

> RQ1: Which of the offered algorithms for malware detection is most suited for detecting malware in real-world apps?

Answering this question assists in the selection of the optimal model for malware detection in Android apps. In this paper, we examine ten methods of feature selection and machine learning algorithms in order to build a new model. Furthermore, we consider various performance-related parameters (accuracy and the F-measure in the case of supervised, semisupervised, and hybrid machine learning algorithms; and intracluster and intercluster distances in the case of unsupervised machine learning algorithms) to determine the model that is best suited for malware detection.

> RQ2: Is the proposed malware detection technique applicable to Android devices? The goal of this research is to determine how effective our malware detection technique is. To do so, we compare its performance with prevalent approaches in the literature.
>
> RQ3: Is a subset of characteristics better than all retrieved features in determining if an app is dangerous?

The purpose of this inquiry is to assess the performance metrics and examine the link between benign and malicious applications. Various feature reduction techniques are studied to identify a subset of traits that are capable of detecting malicious apps.

> RQ4: Which of the implemented feature ranking algorithms performs the best in terms of distinguishing between malicious and benign Android apps?

The performance of machine learning algorithms in terms of feature ranking is influenced by the features and the type of malware data collected. Multiple techniques with diverse criteria have been developed to rank the gathered feature sets. We use several performance criteria to compare the performance of the implemented feature ranking techniques.

> RQ5: Which strategy to select a feature subset outperforms the others in terms of identifying malware in Android apps?

We investigated methods for selecting a feature subset to identify the subset that is most appropriate for detecting malicious Android apps. We weighted the approaches based on F-measure and accuracy for supervised, semisupervised, and hybrid machine learning algorithms, and on intracluster and intercluster distances in the case of unsupervised machine learning algorithms.

> (RQ6: How do the techniques for feature ranking and feature subset selection compare?

The pairwise t-test was used to determine whether the techniques for selecting feature subsets were better than those for feature ranking or whether they performed similarly well.

> RQ7: Do techniques of feature selection have an impact on the outcomes of machine learning-based approaches?

A variety of approaches to feature selection work well with particular machine learning algorithms. This study assesses such techniques by using a diversity of machine learning-based approaches to gauge their effectiveness.

3. Methodology

To ensure clarity and direction in this section, we first discuss the preliminaries and definitions. Second, we discuss the methodology, which is subdivided into two sections: the flowchart and the proposed methodology (framework), respectively. In the flowchart subsection, we discuss the sequential flow of the proposed technique, which starts with task 4 as stated in Figure1. In the proposed methodology (framework) subsection, we discuss the experimental setup, datasets, structure, and pseudocode.

3.1. Preliminaries and Definition. In this section, we go over the basic definitions of the performance-related parameters used to evaluate our proposed technique for selecting feature subsets and for malware detection. All factors were calculated by using a confusion matrix that was composed of classification-related information extracted by using the two techniques. (Table2)

3.1.1. Accuracy. Accuracy is defined as the corrected prediction of malware-infected apps with respect to the total number of benign and malware-infected apps. For supervised, semisupervised, and hybrid machine learning techniques, it is given by

$$Accuracy = \frac{a+d}{N\text{classes}}, \tag{1}$$

where $N\text{classes} = a + b + c + d$, and

$$d = N\text{Benign} \longrightarrow \text{Benign}. \tag{2}$$

3.1.2. F-Measure. We use several machine learning methods to create the feature subset and models of malware detection. Comparing models with high recall and low accuracy, or vice versa, thus becomes challenging. We compared by using the F-measure, which is useful for measuring precision and recall at the same time.

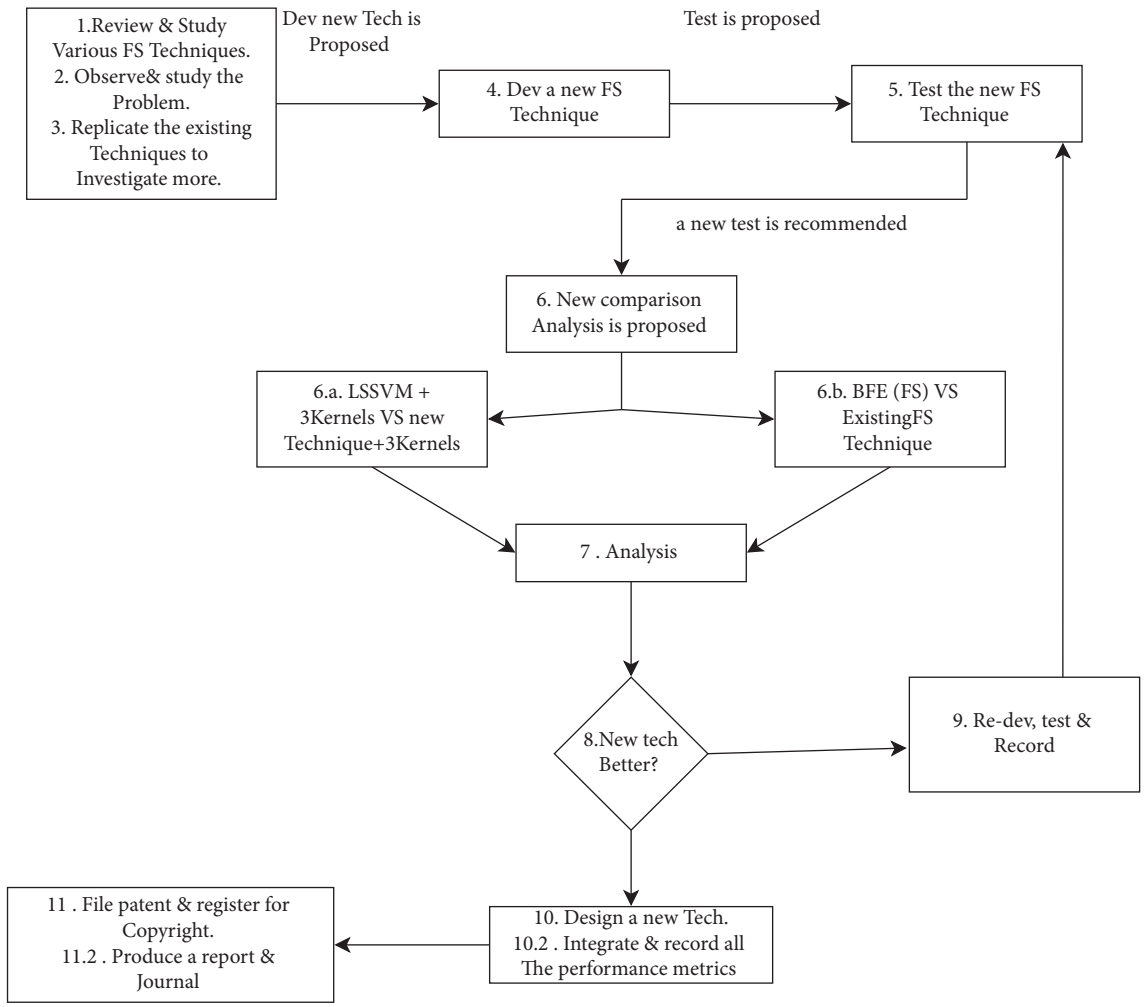

Figure 1: The proposed feature selection-based malware detection technique flowchart.

Table 2:

Dataset abbreviations used	Corresponding names	References
DS1	Dataset partition 1	Mahindru and Sangal [1]
DS2	Dataset partition 2	Mahindru and Sangal [1]
DS3	Dataset partition 3	Mahindru and Sangal [1]
DS4	Dataset partition 4	Mahindru and Sangal [1]
DS5	Dataset partition 5	Mahindru and Sangal [1]

F-measure is defined by

$$F - \text{measure} = 2 * \text{Precision} * \frac{\text{Recall}}{\text{Precision}} + \text{Recall} = \frac{2 * a}{2 * a} + b + c. \quad (3)$$

3.1.3. The Weighted FM. The weighted FM combined measure is defined as follows:

$$\text{FM} = 2 * \text{recall} * \text{precision recall} + \text{precision}, \quad W - \text{FM} = (Fm.Nm) + (Fb.Nb)Nm + Nb, \quad (4)$$

where Fb and Fm represent the FM of the benign and malware datasets, respectively, and Nb and Nm represent the number of samples in the benign and malware datasets. All experiments presented used the 5-fold cross-validation method.

3.2. Methodology

3.2.1. Flowchart. In this section, we discuss the flowchart of our technique. This flowchart depicts the sequential flow of coordinates to ensure clarity, efficiency improvement, long-term effective analysis, and problem-solving. First, we review and study various approaches for feature subset selection purposes, observe and study the problem, and replicate the existing approach to investigate more. Second, we developed

a new embedded feature subset selection technique (embedded BFEDroid). This is where our main contribution to this research starts. Third, we test the new embedded technique. Fourth, we proposed a new comparison analysis. Fifth, we compare the LSSVM +3 kernels + existing feature subset selection model to the newly proposed model LSSVM +3 kernels + embedded feature subset selection technique (BFEDroid). Sixth, we compare and analyze the embedded BFE feature subset selection model with the existing feature subset selection model. Seventh, a newly proposed technique (BFEDroid) is developed if the technique satisfies 7 of our research goals. Lastly, we develop and record the newly proposed technique (BFEDroid). The flowchart is shown in Figure 1.

3.2.2. Proposed Methodology (Framework). We propose an embedded analysis for the detection of malware using a new feature subset selection technique (BFEDroid). The proposed embedded BFEDroid detection technique is paired with the best among the feature ranking techniques (principal component analysis (FR6)), the LSSVM, and the radial basis function kernel (best among the three kernels). This proposed malware detection technique can be updated by retraining on new applications with true labels. We first performed the analysis of LSSVM with 3 kernels using the prevalent feature subset selection and found out the possibilities and limitations of this technique, and then introduced a new technique. This proposed technique uses LSSVM radial basis function with embedded feature subset selection (BFEDroid) (FS) and principal component analysis (FR6) (best among feature ranking techniques) to analyze and overcome the limitations found in the research gap. Note, among these 3 kernels, we used radial basis function (best among the kernels), BFEDroid (FS), and principal component analysis (best among feature ranking techniques) (FR6) in the proposed detection technique.

The permissions and API calls were extracted by using an emulator (Android Studio). It offers the same emulator and offers the same API level and execution environment as mobile phones in the flowchart task number (5). We used the Android system version 6.0 Marshmallow (i.e., API level 23) to extract permissions and API calls from Android apps and construct our dataset for testing. Previous developed frameworks or approaches used the previous version of Android to extract features from them. We chose this particular version of Android for two reasons: it initially requests the user to revoke or allow the permission to use smart phone resources, and then covers 28.1% of Android gadgets, which is higher than other versions of Market23, is very effective in battery life with "deep sleep," and provides App Permission Management update, which is very critical in sustainability. In recent feature-based detection research [1], this Android studio version faired very well in terms of accuracy, precision, and F-measure when embedded with other classifiers and feature subset selection techniques. In the follow-up study [37, 38], this version still performed well with other feature-based detection techniques in terms of accuracy, precision, and F-measures. Following this, if the SDK version is older than 23, the Android prompts the user to grant or remove permission for the app. If the user grants the permission, the call instruction is executed; otherwise, it is not, and other versions can increase runtime and memory usage. This feature has been unavailable in past versions of Android. The detailed framework is represented in Figure 2.

Dataset. We collected datasets from the Mendeley repository.

Dataset Structure. We extracted features from Android apps. In the first phase, to extract features from collected. apk, and extract permissions and API calls from them and save them into the.csv file. $D1$, $D2$, . . ., Dn partitions, the datasets are divided into $D1 = 1000$, $D2 = 1000$, . . ., $Dn = 1000$ partitions.

Our contribution focuses on number four (4) in our methodology flowchart in Figure 1. We propose a new feature selection-based detection technique that segregates the training data for two purposes, namely, the feature ranking module and the feature subset selection module. Only a few subset features will be selected to proceed with the normalization process together with the output from the feature subset selection and feature ranking modules. The selection is based on the embedded BFEDroid (FS). Our proposed framework technique is represented in Figure 2.

In the proposed feature selection-based malware detection technique framework (Figure 2), we combined the least-square support vector machine (LSSVM), radial basis function (the best among the kernels) with an embedded BFEDroid (backward, forward, and exhaustive), and principal component analysis (the best among the feature ranking techniques) to select the best feature set for the best detection module. We discuss the pseudocode of the proposed technique, embedded BFEDroid, as follows: embedded BFE feature selection (EBFE).

Step 1: Choose a significance level (e.g., SL = 0.05 with 95% confidence).

Step 2: Fit a full model including all features.

Step 3: Consider the feature with the average attribute value. If the attribute value > significance level, then go to step 4; otherwise, terminate the process.

Step 4: Remove the feature that is under consideration.

Step 5: Fit a model without this feature. Repeat the entire process from step 3.

(1) LSSVM Classifier. RBF kernel (radial basis function), the hyperparameter denotes γ, the kernel parameter denotes σ, and the test.

Set performances of the binary LSSVM classifier are estimated using the following steps:

Input: selected feature set using EBFE ()

Output: optimal classified set.

Step 1. Set aside 2/3 of the data for the training/validation set and the remaining 1/3 for testing.

Step 2. Starting from $i = 0$, perform 10-fold cross-validation on the training/validation data for each (σ, γ)

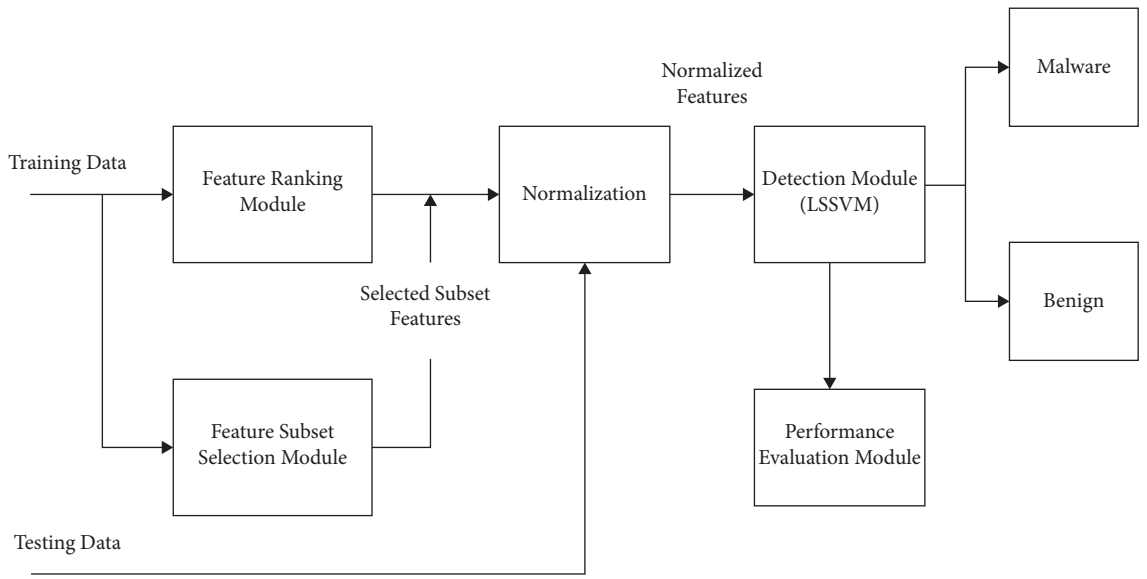

Figure 2: The proposed feature selection-based malware detection technique framework.

Table 3: LSSVM polynomial kernel (classifier) with rough set analysis (FS4) and principal component analysis (PCA) (FR6) (detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.1137	0.89	**0.956**	0.89	0.86	0.93	0.133	0.142	0.8612	1416
DS2	0.1023	0.90	**0.624**	0.90	0.88	0.94	0.133	0.114	0.8832	1540
DS3	0.128	0.88	**1.265**	0.88	0.86	0.93	0.133	0.142	0.8659	1745
DS4	0.0561	0.94	**1.384**	0.94	0.90	0.92	0.142	0.153	0.9022	1950
DS5	0.0745	0.93	**1.748**	0.93	0.92	0.95	0.142	0.153	0.9265	2080

Table 4: LSSVM RBF kernel (classifier) with rough set analysis (FS4) and principal component analysis (PCA) (FR6) (detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.0458	0.95	**0.465**	0.95	0.92	0.93	0.108	0.0625	0.9234	1350
DS2	0.0275	0.93	**0.578**	0.93	0.90	0.90	0.111	0.090	0.9066	1471
DS3	0.047	0.95	**0.873**	0.95	0.93	0.934	0.075	0.0654	0.9372	1621
DS4	0.022	0.98	**1.218**	0.98	0.98	0.967	0.050	0.0322	0.9831	1857
DS5	0.022	0.98	**1.487**	0.98	0.982	0.967	0.049	0.029	0.9825	1959

Table 5: LSSVM linear kernel (classifier) with embedded BFEDroid (FS) and principal component analysis (PCA) (FR6) (detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.0427	0.95	**0.4920**	0.95	0.94	0.952	0.088	0.047	0.9421	1202
DS2	0.059	0.94	**0.5643**	0.94	0.92	0.937	0.111	0.062	0.9274	1378
DS3	0.045	0.95	**0.895**	0.95	0.94	0.952	0.081	0.047	0.9469	1587
DS4	0.018	0.98	**1.200**	0.98	0.97	0.975	0.038	0.024	0.9784	1696
DS5	0.0289	0.97	**1.360**	0.97	0.98	0.983	0.025	0.016	0.9815	1811

Table 6: LSSVM polynomial kernel (classifier) with embedded BFEDroid (FS) and principal component analysis (PCA) (FR6) (detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.0679	0.93	**0.723**	0.93	0.93	0.913	0.857	0.015	0.9304	1110
DS2	0.0482	0.95	**0.456**	0.95	0.95	0.955	0.6	0.029	0.9515	1250
DS3	0.070	0.93	**0.965**	0.93	0.93	0.913	0.857	0.015	0.9382	1487
DS4	0.0464	0.95	**1.100**	0.95	0.95	0.955	0.6	0.029	0.9534	1612
DS5	0.0682	0.93	**1.570**	0.93	0.93	0.913	0.857	0.015	0.9323	1750

Table 7: LSSVM RBF kernel (classifier) l with embedded BFEDroid (FS) and principal component analysis (PCA) (FR6) (proposed detection technique).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy	Memory
DS1	0.0168	0.98	**0.325**	0.98	0.96	0.961	0.041	0.038	0.9642	1040
DS2	0.0275	0.97	**0.492**	0.97	0.94	0.943	0.068	0.056	0.9446	1154
DS3	0.018	0.98	**0.754**	0.98	0.97	0.970	0.030	0.029	0.9714	1396
DS4	0.010	0.99	**1.159**	0.99	0.99	0.990	0.010	0.009	0.9923	1540
DS5	0.010	0.99	**1.264**	0.99	0.99	0.990	0.010	0.009	0.9945	1680

Table 8: Forward subset selection (feature selection technique) (FS).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy
DS1	0.0178	0.92	0.589	0.98	0.98	0.970	0.030	0.009	0.9833
DS2	0.1045	0.91	0.698	0.97	0.90	0.884	0.139	0.065	0.9021
DS3	0.0600	0.89	0.952	0.98	0.94	0.923	0.128	0.016	0.9452
DS4	0.0100	0.95	1.312	0.99	0.93	0.967	0.057	0.076	0.9332
DS5	0.0100	0.95	1.471	0.99	0.95	0.960	0.066	0.04	0.9556

Table 9: Backward subset selection (feature selection technique) (FS).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy
DS1	0.047	0.89	0.362	0.89	0.95	0.943	0.0625	0.038	0.9544
DS2	0.070	0.94	0.454	0.94	0.93	0.909	0.104	0.038	0.9384
DS3	0.073	0.93	0.787	0.93	0.92	0.961	0.0454	0.107	0.9239
DS4	0.090	0.96	1.265	0.96	0.91	0.943	0.0566	0.107	0.9123
DS5	0.020	0.97	1.317	0.97	0.98	0.980	0.020	0.019	0.9876

Table 10: Exhaustive subset selection (feature selection technique) (FS).

Datasets	ERR	FM	Running time (SI)	WFM	AUC	Precision	FPR	FNR	Accuracy
DS1	0.0124	0.96	**0.258**	0.96	0.99	0.990	0.010	0.009	0.9942
DS2	0.0425	0.95	**0.352**	0.95	0.96	0.952	0.051	0.029	0.9646
DS3	0.0587	0.95	**0.524**	0.95	0.94	0.925	0.083	0.038	0.9414
DS4	0.0100	0.94	**1.092**	0.94	0.99	0.990	0.010	0.009	0.9923
DS5	0.0100	0.99	**1.035**	0.99	0.99	0.990	0.010	0.009	0.9945

Table 11: Feature ranking techniques on different datasets (F-measure) (FR).

Datasets	FR1	FR2	FR3	FR4	FR5	FR6
DS1	87	90	89	91	97	98
DS2	85	88	89	85	93	94
DS3	85	90	92	89.4	96	97.5
DS4	82	84	86.6	89.7	85	96.2
DS5	80	85	89	91.2	93	94

Table 12: Feature ranking techniques on different datasets (accuracy) (FR).

Datasets	FR1	FR2	FR3	FR4	FR5	FR6
DS1	85	89.4	87.2	90.6	95	96
DS2	83	86	87	83	91	92
DS3	83.5	87	90	88	93	95.8
DS4	80	81	85.3	87	83	94
DS5	78	83	87	89	91	92

combination of the initial candidate tuning sets $\sum 0 = \{0.5, 5, 10, 15, 25, 50, 100, 250, 500\} \ldots \sqrt{n}$ and $\tau 0 = \{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}$.

Step 3. Choose the optimal (σ, γ) from the tuning sets $\sum i$ and τi by looking at the best cross-validation performance for each (σ, γ) combination.

Step 4. If $i = i_{\max}$, go to Step 5; else $i := i + 1$, construct a locally refined grid.

$\sum i \times \tau i$ around the optimal hyperparameters (σ, γ) and go to Step 3.

Step 5. Construct the LSSVM classifier using the total training/validation set for the optimal choice of the tuned hyperparameters (σ, γ).

Step 6. Assess the test set accuracy by means of the independent test set.

We explain the embedded BFEDroid literally as follows:

(i) First, the best single feature is selected (i.e., using some criterion function).

Table 13: Prevalent feature subset selection techniques on different datasets (accuracy and precision) (FS).

Datasets	FS1 (acc)	FS2 (acc)	FS3 (acc)	FS4 (acc)	FS1 (prec)	FS2 (prec)	FS3 (prec)	FS4 (prec)
DS1	94.5	97.3	94.9	96.8	92.0	91.7	92.7	93
DS2	94.5	97.6	96.2	97.5	84.4	85.9	90	90
DS3	92.1	84.9	83	90.7	88.3	91.2	91	93.4
DS4	87.6	84.8	85	89.7	94.7	91.4	96	96.7
DS5	93	95.7	92	96.5	94.0	94.0	95.5	96.7

Table 14: Prevalent feature subset selection techniques on different datasets (F-measure and WFM) (FS).

Datasets	FS1 (F-M)	FS2 (F-M)	FS3 (F-M)	FS4 (F-M)	FS1 (WFM)	FS2 (WFM)	FS3 (WFM)	FS4 (WFM)
DS1	95.3	98.3	95.3	95	95	94	93.4	95
DS2	95.3	96.6	98.2	93	93	**95**	95	96
DS3	94.7	87.9	88	95	95	88	86	96
DS4	89.4	85.9	80	98	98	87	90	97
DS5	95.1	96.7	94	98	98	93	94	97

Table 15: Prevalent feature subset selection techniques on different datasets (error ratio and AUC) (FS).

Datasets	FS1 (ERR)	FS2 (ERR)	FS3 (ERR)	FS4 (ERR)	FS1 (AUC)	FS2 (AUC)	FS3 (AUC	FS4 (AUC)
DS1	5.5	2.7	5.1	3.2	94.5	97.3	94.9	96.8
DS2	5.5	2.4	3.8	2.5	94.5	97.6	96.2	97.5
DS3	7.9	15.1	17	9.3	92.1	84.9	83	90.7
DS4	12.4	15.2	15	10.3	87.6	84.8	85	89.7
DS5	7.0	4.3	8	3.5	93	95.7	92	96.5

Table 16: Feature selection technique (FS) compared with proposed and proposed individual techniques (accuracy and precision).

Datasets	FS1 (acc)	FS2 (acc)	FS3 (acc)	BFE (acc)	FS1 (prec)	FS2 (prec)	FS3 (prec)	BFE (prec)
DS1	95	99	97.8	99	97.0	94.3	95.7	99
DS2	96.2	99.2	98.9	99.3	88.4	90.9	91	95.2
DS3	93	90	86	94.5	92.3	96.1	92	92.5
DS4	90	88	90	92.6	96.7	94.3	97	99
DS5	93	97	95	99.4	96	98	98.5	99

Table 17: Feature selection technique (FS) compared with proposed and proposed individual techniques (F-measure and WFM).

Datasets	FS1 (FM)	FS2 (FM)	FS3 (FM)	BFE (FM)	FS1 (WFM)	FS2 (WFM)	FS3 (WFM)	BFE (WFM)
DS1	98	98	98.8	98	98	98	98.8	98
DS2	97	99	99	97	97	99	99	97
DS3	95	91	89	98	95	91	89	98
DS4	92	92	94	99	92	92	94	99
DS5	96	99	98	99	96	99	98	99

Table 18: Feature selection technique (FS) compared with proposed and proposed individual techniques (error rate ratio and AUC).

Datasets	FS1 (err)	FS2 (err)	FS3 (err)	BFE (err)	FS1 (AUC)	FS2 (AUC)	FS3 (AUC)	BFE (AUC)
DS1	5.0	1.0	2.2	1.0	91	93.4	95.2	96
DS2	3.8	0.8	1.1	0.7	97	92	92	95
DS3	7.0	10	14.1	5.5	96	94	90	97
DS4	10.0	12	10.0	7.4	90	94	96	99
DS5	7.0	3	5	0.6	97	99.2	98.4	99

Table 19: Prevalent and proposed feature subset selection technique (FS) comparison based on runtime (SI).

Datasets	Correlation best feature selection	Classifier subset evaluation	Filtered subset evaluation	Rough set analysis (RSA)	Forward_ F-selection	Backward F-selection	Exhaustive F-selection	Proposed BFE
DS1	0.952	0.891	0.873	0.712	0.299	0.181	0.143	0.115
DS2	1.432	1.058	0.987	0.932	0.358	0.254	0.172	0.162
DS3	1.543	1.342	1.243	1.132	0.506	0.397	0.264	0.211
DS4	1.675	1.598	1.521	1.514	0.659	0.625	0.521	0.494
DS5	1.854	1.791	1.725	1.685	0.714	0.661	0.514	0.456

Table 20: Prevalent and proposed feature subset selection technique (FS) comparison based on memory.

Datasets	Correlation best feature selection	Classifier subset evaluation	Filtered subset evaluation	Rough set analysis (RSA)	Forward F-selection	Backward F-selection	Exhaustive F-selection	Proposed BFE
DS1	987	859	654	542	312	285	250	208
DS2	1032	913	784	648	580	498	365	280
DS3	1243	1100	1032	845	716	652	547	489
DS4	1375	1259	1105	1051	956	845	685	610
DS5	1554	1371	1272	1165	1071	961	821	786

(ii) Then, pairs of features are formed using one of the remaining features and this best feature, and the best pair is selected.

(iii) Next, triplets of features are formed using one of the remaining features and these two best features, and the best triplet is selected and a contiguous number of features (n) are selected.

(iv) Then, each feature is deleted one at a time, the criterion function is computed for all subsets with n-1 features, and the worst feature is discarded.

(v) Next, each feature among the remaining n-1 is deleted one at a time, and the worst feature is discarded to form a subset with n-2 features.

(vi) This procedure continues until a predefined number of features are left.

The feature selection technique's core goal is to find the feature subset that provides the best ability to discriminate across various feature subset groups. In many circumstances, including all characteristics in a classifier does not result in the optimum performance. Feature selection also assists users in gaining better knowledge of qualities that are critical to diagnosing the data of interest.

The forward feature selection procedure begins by evaluating all feature subsets that contain only one input attribute. Forward selection finds the best subset consisting of two components, $X(1)$, and one other feature from the remaining M-1 input attributes. Hence, there are a total of M1 pairs. Let us assume that $X(2)$ is the other attribute in the best pair besides $X(1)$. Input subsets with three, four, or more features are subsequently evaluated. According to forward selection, the best subset with m features is the m-tuple consisting of $X(1)$, $X(2)$, ..., $X(m)$, while the best feature set overall is the winner of all M steps.

The LSSVM is a supervised machine learning technique that has been applied to a variety of domains, including regression, classification, and outlier identification. The essential concept of the SSVM involves two-class issues, in which data are sorted into classes based on the optimal hyperplane generated by support vectors. The support vectors between classes determine the boundary of the training set. We used the LSSVM with unique kernels as a classifier to develop a model that can distinguish between benign and malicious applications.

4. Results and Discussion

In this section, we evaluate our newly proposed (BFEDroid) feature selection-based malware detection technique against the prevalent techniques. The seven core goals of our experiments were to develop a detection technique that could decrease the execution time and memory usage, and improve the FPR, FNR, precision rate, and detection rate of resilience to obfuscation. Highlights of the evaluation are as follows:

(a) Tables 1, 3, and 4 list the detection empirical comparative results of prevalent detection techniques for different categories of Android apps by considering the LSSVM with three distinct kernel functions, i.e., linear, RBF, and polynomial functions. The detection technique with LSSVM, radial basis function kernel (RBF), principal component analysis (FR6), and rough set analysis (FS4) outperformed other prevalent detection techniques. Tables 1, 3, and 4 depict the detection results of LSSVM with each kernel with rough set analysis subset selection and principal component analysis (FR6). Tables 1, 3, and 4 depict the detection

Table 21: Proposed technique compared with other feature-based detection techniques (detection comparative results).

References	Detection techniques	Feature selection techniques	Platform	ERR	Runtime (SI)	WFM (%)	AUC (%)	Prec	FPR	FNR	Acc (%)	Memory	FM (%)
Mahindru and Sangal [39]	Deep learning model (DNN) with PARU	PARU (permissions, API calls, rating of an app, and users download the app). Self-written algorithm	Android	*x*	*x*	93	*x*	*x*	*x*	*x*	98.9	*x*	93
Taheri et al. [40]	LightGBM with fuzzy	Fuzzy C-means algorithm	Android	*x*	*x*	*x*	98.74	97.70	*x*	*x*	94.63	*x*	90.37
Mahindru and Sangal [41]	HybriDroid	Feature selection validation algorithm	Android	*x*	14	*x*	98	*x*	*x*	*x*	98.8	*x*	*x*
Mahindru and Sangal [38]	SOMDROID	Principal component analysis (PCA) feature selection algorithm	Android	*x*	*x*	*x*	*x*	*x*	*x*	*x*	92.4	*x*	91
Proposed technique	BFEDroid	Embedded BFE algorithm	Android	0.01	1.264	99	99	99	0.010	0.009	99.45	1680	99

Table 22:

Matrix abbreviations used	Corresponding names
ERR	Error rate ratio
F-M	F-measure/score
AUC	Area under the ROC curve
FPR	False-positive ratio
FNR	False-negative ratio
W-FM	Weighted F-measure

Table 23:

Classifier abbreviations used	Corresponding names
RBF	Radial basis function
LSSVM	Least-square support vector machine

performance of each LSSVM kernel (linear, polynomial, and radial basis function) with principal

Table 24:

Existing feature selection approach abbreviations used	Corresponding names	References
FS1	Correlation best feature selection	Akram et al. [42]
FS2	Classifier subset evaluation	Fiky [43]
FS3	Filtered subset evaluation	Shafiq et al. [44]
FS4	Rough set analysis (RSA)	Mahindru and Sangal [1]

Table 25:

Proposed feature selection approach abbreviations used	Corresponding names	References
FS1	Forward feature subset selection	Fiky [43]
FS2	Backward feature subset selection	Fiky [43]
FS3	Exhaustive feature subset selection	Fiky [43]
FS4	Embedded (FS1, FS2, and FS3) BFE feature subset selection	Fiky [43]

Table 26:

Feature ranking approach abbreviations used	Corresponding names	References
FR1	Chi-square test	Nassar et al. [45]
FR2	Gain ratio feature evaluation	Karegowda et al. [46]
FR3	Filtered subset evaluation	Vinutha and Poornima [47]
FR4	Information gain feature evaluation	Omuya et al. [48] and vinutha and Poornima [47]
FR5	Logistic regression analysis	Abawajy et al. [49]
FR6	Principal component analysis (PCA)	Omuya et al. [48]

Table 27:

Feature subset selection technique (prevalent) abbreviations used	Corresponding names	References
FS1	Correlation best feature selection	Mahindru and Sangal [1]
FS2	Classifier subset evaluation	Mahindru and Sangal [1]
FS3	Filtered subset evaluation	Mahindru and Sangal [1]
FS4	Rough set analysis (RSA)	Mahindru and Sangal [1]

Table 28:

Feature subset selection technique (proposed) abbreviations used	Corresponding names	References
FS1	Forward feature subset selection	Wen and Chow [29]
FS2	Backward feature subset selection	Wen and Chow [29]
FS3	Exhaustive feature subset selection	Wu and Kanai [30]
BFE	Embedded technique (backward + forward + exhaustive)	Our proposed technique

component analysis (i.e., best functional ranking technique) and rough set analysis (best among the existing feature subset selection).

(b) Tables 5–7 depict the detection empirical comparative results using the proposed BFE feature selection technique (FS) on different categories of Android apps by considering the LSSVM with three distinct kernel functions, i.e., linear, RBF, and polynomial with principal component analysis (FR6). Note that the proposed detection technique is composed of LSSVM, RBF with principal component analysis (FR6), and BFEDroid (FS). The proposed technique was compared by considering LSSVM with polynomial, RBF, and linear kernels, and each individual LSSVM kernel was embedded with BFEDroid and principal component analysis (PCA) (FR6) (best among feature ranking techniques). The proposed detection technique with embedded BFEDroid (FS) (proposed feature subset selection technique) with LSSVM, radial basis function kernel (RBF), and principal component analysis (FR6) outperforms others. Tables 5–7 depict the detection performance of the newly proposed BFE with each LSSVM kernel (linear, polynomial, and radial basis functions) with principal component analysis (i.e., best functional ranking technique).

(c) Tables 8–10 show the proposed individual feature subset selection techniques' comparative results for different categories of Android apps by comparing the forward, backward, and exhaustive feature subset selection techniques. Based on the findings, the exhaustive subset selection technique (FS) outperforms other proposed individual (FS) techniques, which make up our proposed embedded model (BFEDroid) as shown below.

(d) Tables 11 and 12 show the feature ranking techniques' (FR) comparative results for different categories of Android apps in terms of the correlation of the best feature and the assessment of subsets obtained by the classifier. The principal component analysis (FR6) outperformed other (FR) techniques as shown below. Tables 13–15 show the existing feature subset selection techniques (FS), comparative results for different categories of Android apps in terms of the correlation of the best features, and the assessment of subsets obtained by the classifier. The rough set analysis (FS4) outperformed other prevalent techniques (FS) as shown below.

(e) Tables 16–18 list the proposed feature subset selection technique (FS) comparative results for different categories of Android apps in terms of forward, backward, exhaustive, and embedded BFE-based subset selection (forward, backward, and exhaustive). The proposed BFE (FS) technique outperforms the other proposed individual techniques (FS), which make up the proposed technique (BFEDroid), which is shown below.

(f) Table 19 compares the prevalent and proposed feature subset selection techniques BFE (FS) in terms of execution time. The proposed embedded BFEDroid (FS) outperforms other (FS) techniques in terms of execution time.

(g) Table 20 compares the prevalent and proposed feature subset selection techniques (FS) in terms of memory consumption. The proposed embedded BFEDroid (FS) outperforms other (FS) techniques in terms of memory consumption.

(h) In this section (Table 21), the proposed feature selection-based detection technique (LSSVM, RBF with principal component analysis (FR6), and BFEDroid (FS)) is compared with other recent feature-based detection techniques. The proposed technique outperformed other detection techniques in terms of memory consumption, runtime, accuracy, precision, AUC, ERR, and others. (Tables22–24)

(A) Tables 1, 3, and 4 list the detection empirical comparative results on prevalent detection techniques for different categories of Android apps by considering the LSSVM with three distinct kernel functions, i.e., linear, RBF, and polynomial functions. The detection technique with LSSVM, radial basis function kernel (RBF), principal component analysis (FR6), and rough set analysis (FS4) outperformed other prevalent detection techniques. Tables 1, 3, and 4 depict the detection results of LSSVM with each kernel with the rough set analysis feature selection technique and principal component analysis (FR6). Tables 1, 3, and 4 depict the detection performance of each LSSVM kernel (linear, polynomial, and radial basis function) with principal component analysis (i.e., best functional ranking technique) and rough set

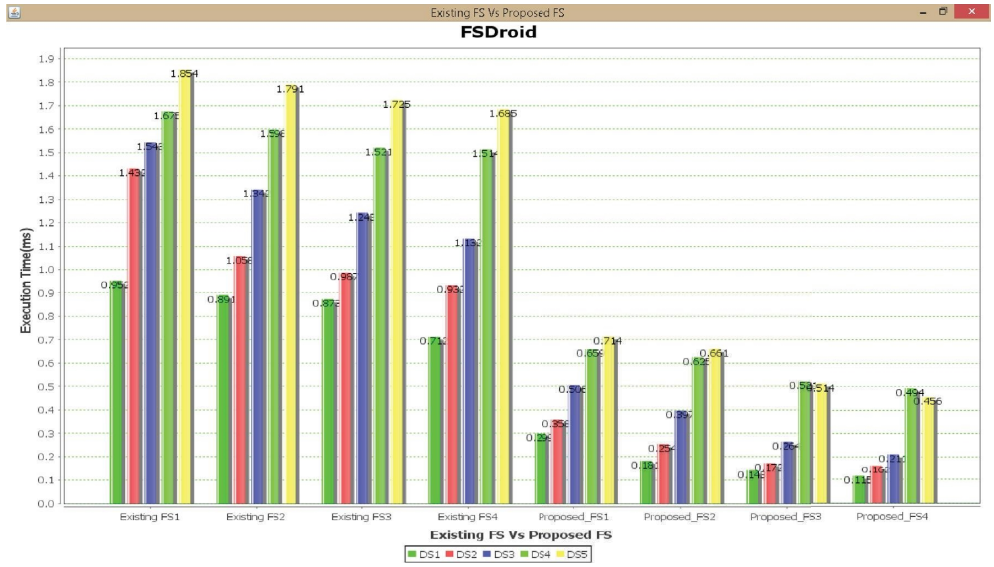

Figure 3: Execution time (SI) comparison of existing feature selection (FS) techniques and proposed BFE (FS) techniques. Proposed FS4 depicts the Embedded BFE (FS4), while other proposed FSs depict the individual units of the BFE (FS).

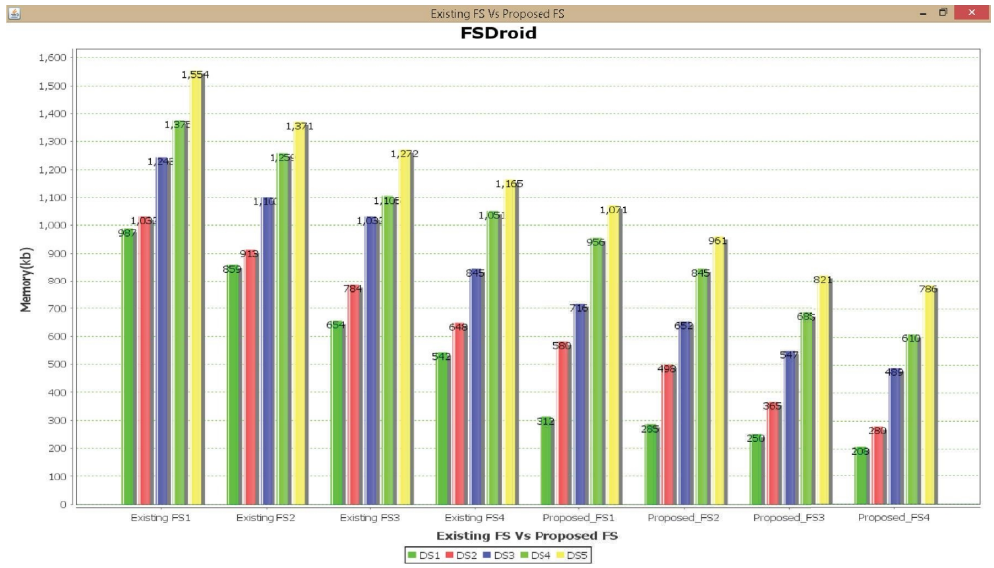

Figure 4: Memory consumption (kb) comparison of existing feature selection (FS) techniques and proposed BFE (FS) techniques. Proposed FS4 depicts the Embedded BFE (FS4), while other proposed FSs depict the individual units of the BFE (FS).

analysis (best among the existing feature subset selection). Figure 5 depicts the accuracy comparison of existing FSDroid detection with rough set analysis (FS) and principal component analysis (FR) on each of the SVM kernels. Figure 6 shows existing malware detection techniques' accuracy, while Figure 7 depicts their respective F-measure.

(B) Tables 5–7 depict the malware detection empirical comparative results using the proposed BFE feature selection technique (FS) on different categories of Android apps by considering the LSSVM with three distinct kernel functions, i.e., linear, RBF, and polynomial with principal component analysis (FR6). Note that the proposed detection technique is composed of LSSVM, RBF with principal component analysis (FR6), and BFEDroid (FS). The proposed technique was compared by considering LSSVM with polynomial, RBF, and linear kernels, and each individual LSSVM kernel was embedded with BFEDroid and principal component analysis (PCA) (FR6) (best among feature ranking techniques). The proposed detection technique with embedded BFEDroid (FS) (proposed feature subset selection technique) with LSSVM, radial basis function kernel (RBF),

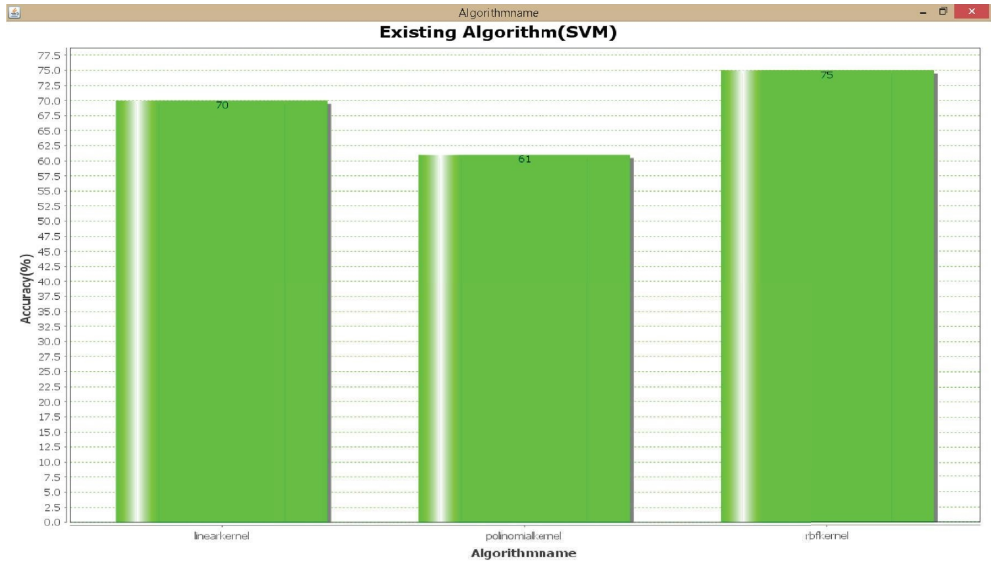

Figure 5: Accuracy comparison of existing FSDroid detection with rough set analysis (FS) and principal component analysis (FR) on each of the SVM kernels.

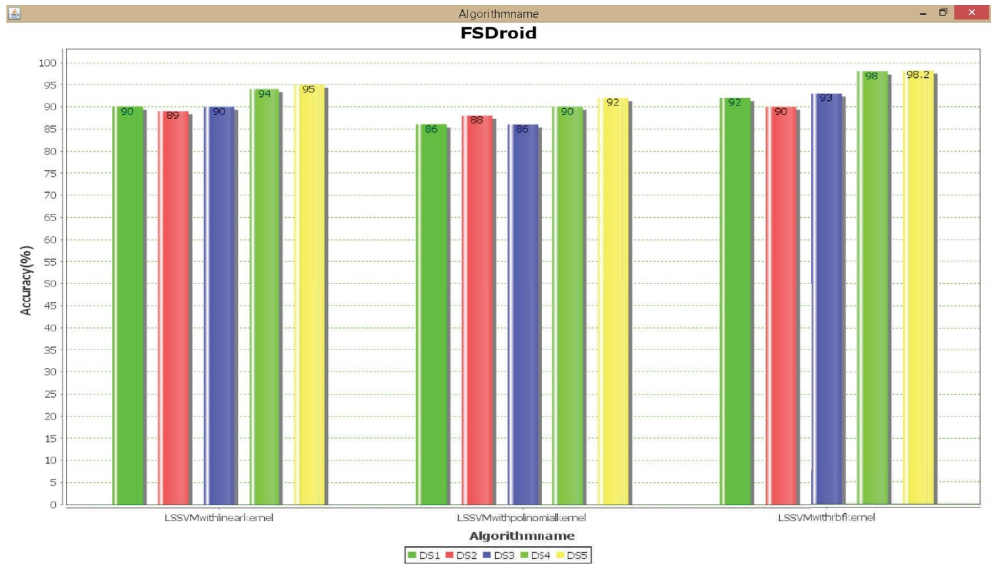

Figure 6: Accuracy comparison of existing FSDroid detection with rough set analysis (FS) and principal component analysis (FR) on each of the LSSVM kernels.

and principal component analysis (FR6) outperforms others. Tables 5–7 depict the detection performance of the newly proposed BFE with each LSSVM kernel (linear, polynomial, and radial basis functions) with principal component analysis (i.e., best functional ranking technique). Figure 3 depicts the execution time (SI) comparison of existing feature selection (FS) techniques and proposed BFE (FS) techniques. Proposed FS4 depicts the embedded BFE (FS4), while other proposed FSs depict the individual units of the BFE (FS). Figure 4 shows a memory consumption (kb) comparison of existing feature selection (FS) techniques and proposed BFE (FS) techniques. Proposed FS4 depicts the embedded BFE (FS4), while other proposed FSs depict the individual units of the BFE (FS). These LSSVM 3 kernels with principal component analysis (FR6) and BFE (FS4) are shown in Figures 10 and 11. These Figures 10 and 11 depict their respective accuracy and F-measure.

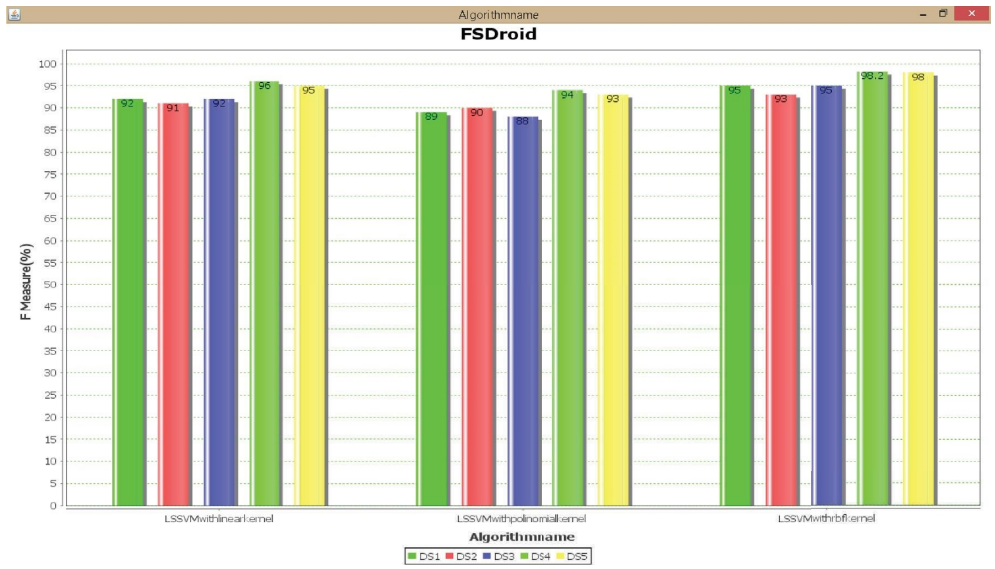

Figure 7: F-measure comparison of existing FSDroid detection with rough set analysis (FS) and principal component analysis (FR) on each of the LSSVM kernels.

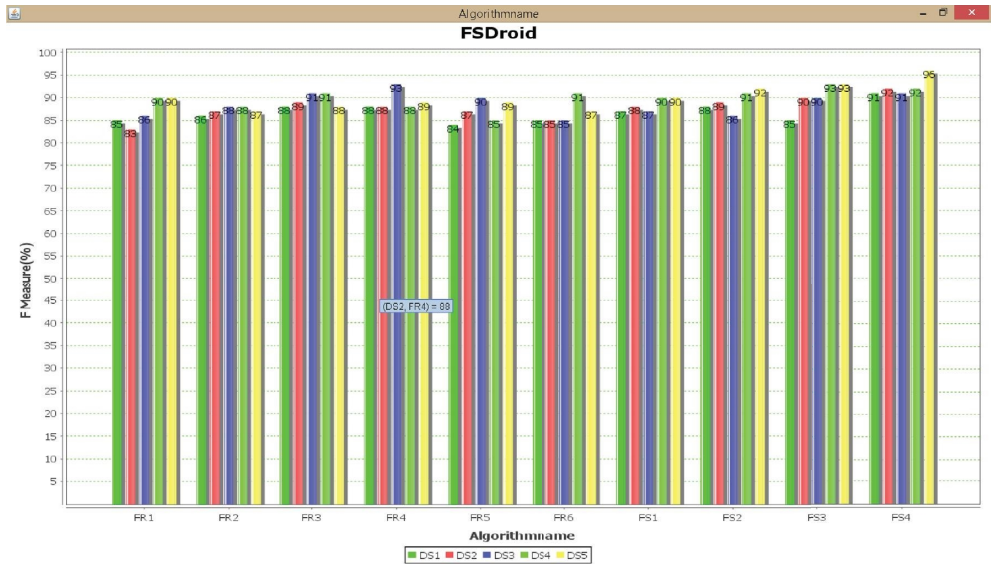

Figure 8: (FR) Droid F-measure comparison for various datasets and existing (FS) feature selection.

(C) Tables 8–10 show the proposed individual feature subset selection techniques' comparative results for different categories of Android apps by comparing the forward, backward, and exhaustive feature subset selection techniques. Based on the findings, exhaustive subset selection (FS) outperforms other proposed individual (FS) techniques, which make up our proposed embedded model (BFE-Droid) as shown below. Figures 12 and 13 depict the accuracy and F-measure results.

(D.1) Tables 11 and 12 compare the feature ranking technique (FR) results for different categories of Android apps in terms of the correlation between the best feature and the classifier's assessment of subsets. The principal component analysis (FR6) outperformed other (FR) techniques as shown below. Figure 8 depicts the respective F-Measure for both the feature ranking techniques and existing feature selection techniques. Figure 9 shows the feature ranking techniques and existing feature selection techniques' accuracy.

(D.2) Tables 13–15 present existing feature subset selection techniques (FS), comparative results for different categories of Android apps in terms of the correlation of the best features, and evaluation of subsets obtained by the classifier.

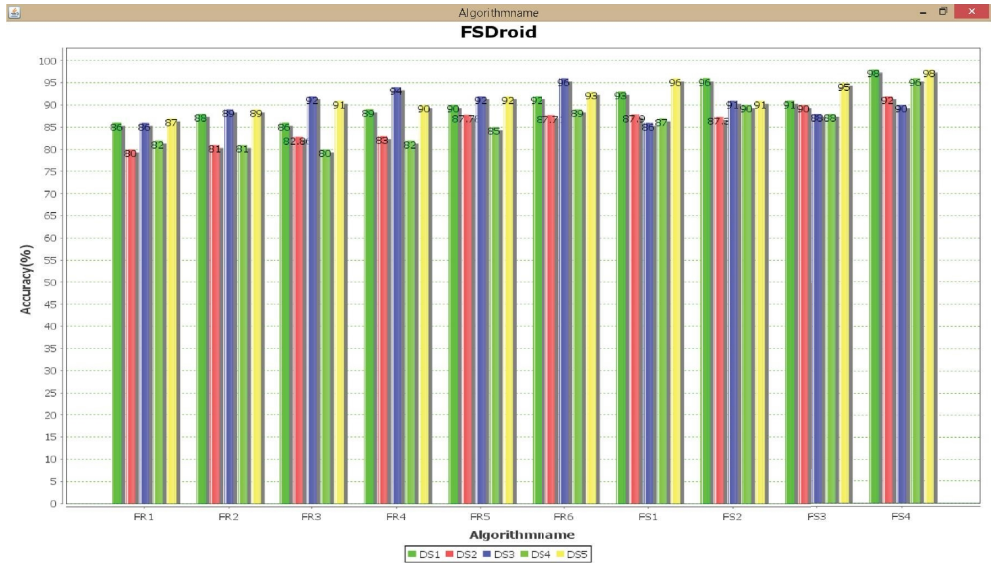

Figure 9: (FR) Droid accuracy comparison for various datasets and existing (FS) feature selection.

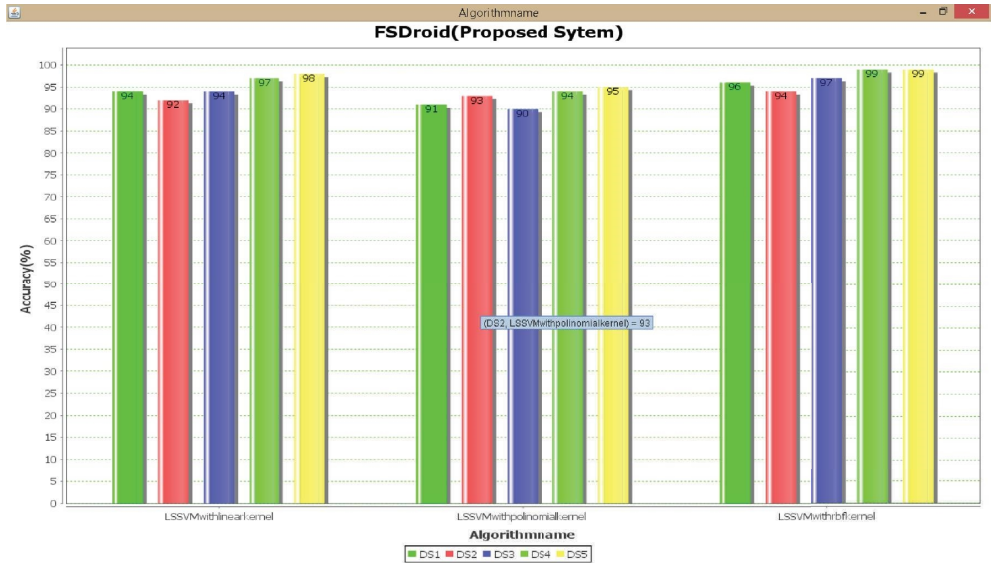

Figure 10: Accuracy comparison of the proposed BFE (FS) Droid with LSSVM different kernels, BFE (FS), and principal component analysis (FR).

The rough set analysis (FS4) outperformed other prevalent techniques (FS), as shown below. Figure 8 depicts the respective F-Measure for both the feature ranking techniques and existing feature selection techniques. Figure 9 shows the feature ranking techniques and existing feature selection techniques' accuracy.

(E) Tables 16–18 list the proposed feature subset selection technique (FS) comparative results for different categories of Android apps in terms of forward, backward, exhaustive, and embedded BFE-based subset selection (forward, backward, and exhaustive). The proposed BFE (FS) technique outperforms the other proposed individual techniques (FS), which make up the proposed technique (BFEDroid), which is shown below. Figure 3 depicts existing Feature selection (FS) techniques, proposed individual BFE units (FS) techniques, and proposed embedded BFE (FS4) techniques' consumption time (SI), while Figure 4 shows existing feature selection (FS) techniques, proposed individual BFE (FS) techniques, and proposed embedded BFE (FS4) techniques' memory consumption (kb).

(F) Table 19 compares the prevalent (FS) and proposed feature subset selection technique, BFE (FS), in terms of execution time. The proposed embedded BFEDroid (FS) outperforms other (FS) techniques in terms of

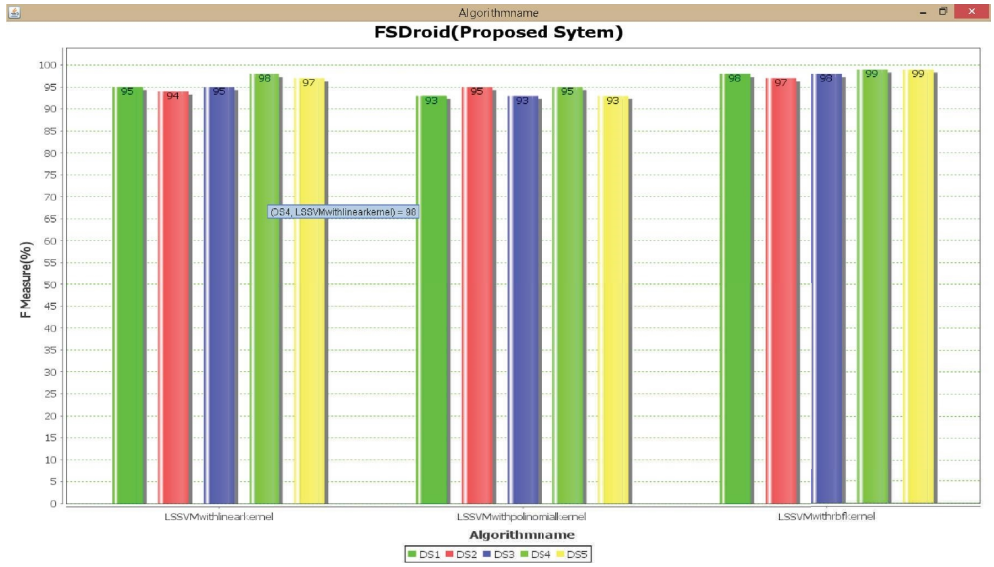

Figure 11: F-measure comparison of the proposed BFE (FS) Droid with LSSVM different kernels, BFE (FS), and principal component analysis (FR).

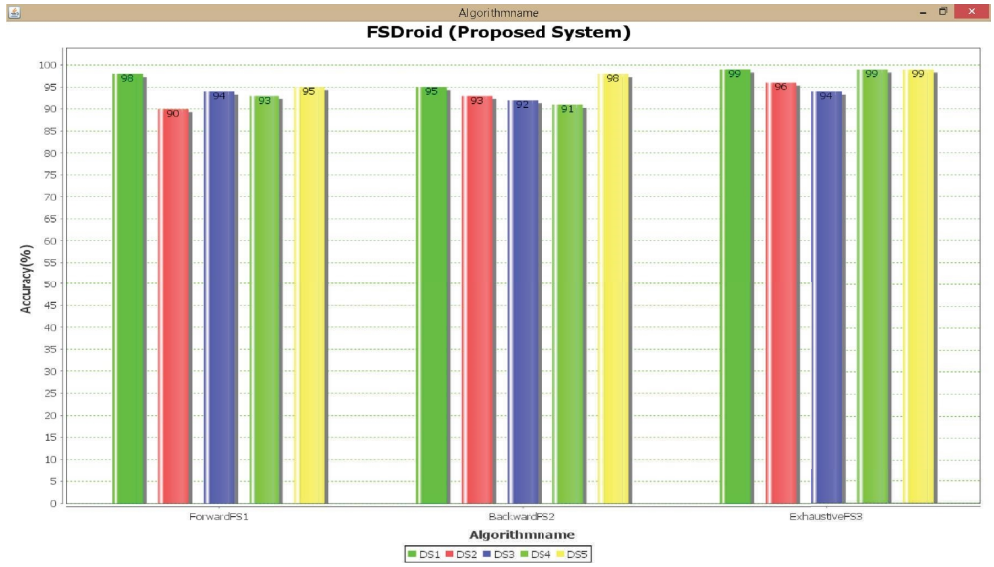

Figure 12: Accuracy comparison of (FS) individual proposed BFE Droid units with forward, backward, and exhaustive subset selection schemes.

execution time. Figure 3 depicts existing Feature selection (FS)techniques, proposed individual BFE units (FS) techniques, and proposed embedded BFE(FS4) techniques' consumption time (SI). Figure 14 shows the execution time comparison of existing and proposed individual BFE unit (FS) techniques. Figure 15 depicts a comparison of memory consumption (kb) between existing and proposed individual BFE units (FS) techniques.

(G) Table 20 compares the prevalent and proposed feature subset selection techniques (FS) in terms of memory consumption. The proposed embedded BFEDroid (FS) outperforms other (FS) techniques in terms of memory consumption (kb). Figure 4 shows existing Feature selection (FS) techniques, proposed individual BFE units (FS) techniques, and proposed embedded BFE (FS4) techniques' memoryconsumption (kb). Tables 8, 9, and 10 show the proposed feature selection technique (FS) for individual units, respectively.

The above comparison, in terms of execution time and memory access, between the prevalent and the proposed feature subset selection technique, (BFE) (FS) shows that the latter was superior on both measures.

Thus, the proposed BFEDroid (FS) with forward, backward, and exhaustive techniques for selecting a feature subset reduces the execution

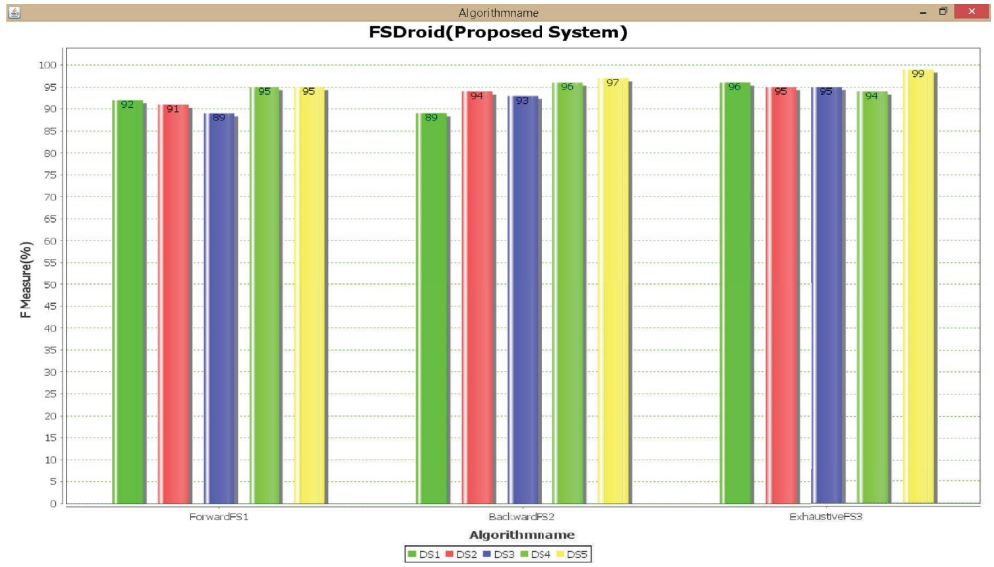

Figure 13: F-measure comparison of individual proposed BFE units (FS) Droid with forward, backward, and exhaustive subset selection schemes.

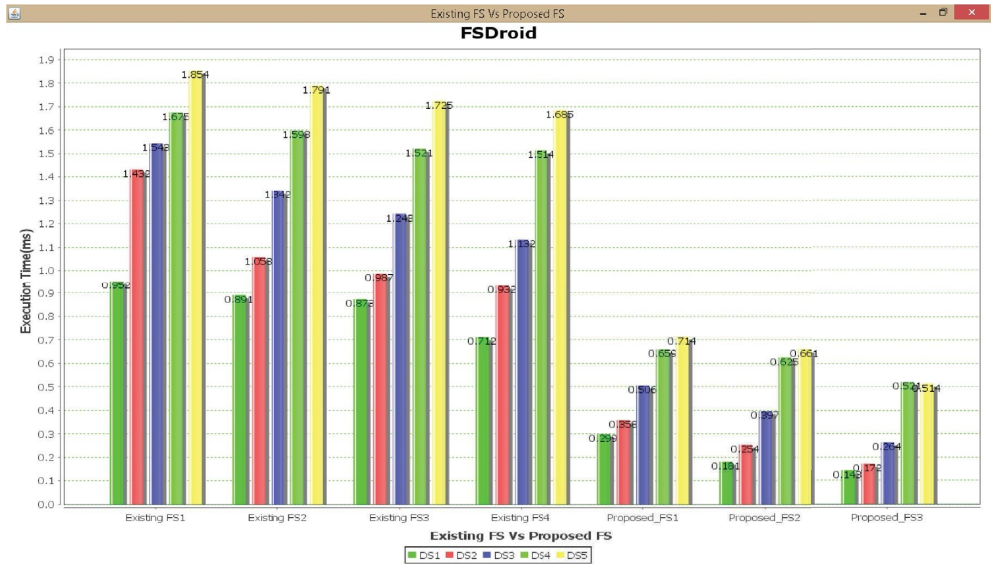

Figure 14: Execution time comparison of existing and proposed individual BFE unit (FS) techniques.

time and memory access, which renders it more efficient than prevalent techniques for malware detection in Android apps.

(H) In this section (Table 21), the proposed feature selection-based detection technique (LSSVM, RBF with principal component analysis (FR6), and BFEDroid (FS)) is compared with other recent feature-based detection techniques. The proposed detection technique outperformed other detection techniques in terms of memory consumption, runtime, accuracy, precision, AUC, ERR, and others. (Tables 25–28)

4.1. Discussion. We now discuss the above findings to determine the best performance among the models considered.Figures3–15.

RQ1: In this study, we evaluated ten feature selection strategies and machine learning algorithms to generate separate models. To identify the model best suited to malware detection, we considered a number of performance-related parameters (accuracy and the F-measure in the case of semisupervised, supervised, and hybrid machine learning algorithms; and these, in addition to intracluster and intercluster distances in the case of unsupervised learning methods, are listed in

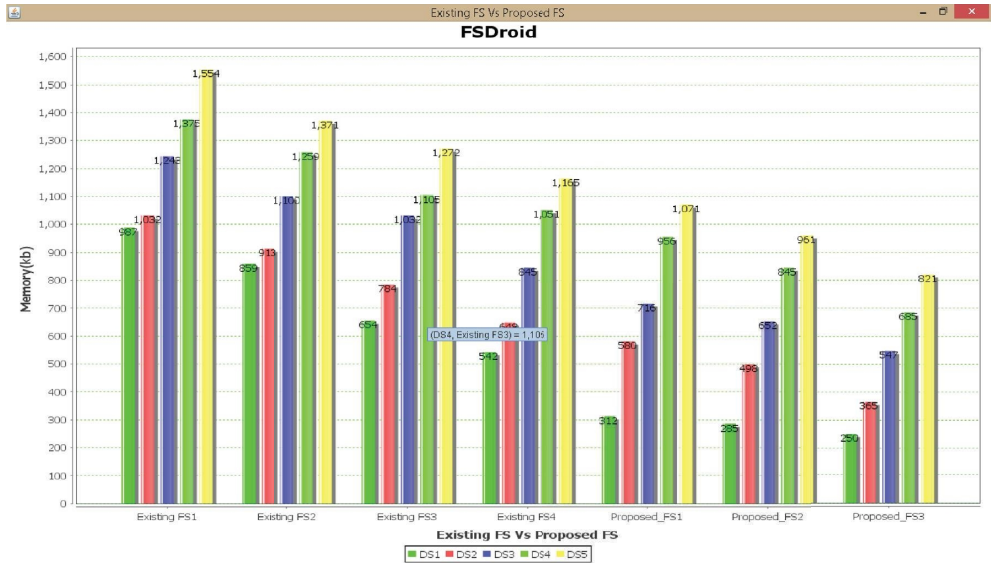

Figure 15: Memory consumption (kb) comparison of existing and proposed individual BFE unit (FS) techniques.

Tables 1, 3–7). In terms of feature ranking techniques, principal component analysis (PCA) (FR) outperformed the other feature ranking techniques (FR), and embedded BFE (FS) outperformed other feature subset selection techniques (FS). For detection purposes, the proposed detection technique (LSSVM, RBF, principal component analysis (FR) with BFE (FS)) outperformed the other detection techniques.

RQ2: This question pertains to the proposed malware detection technique's effectiveness. It is compared with prevalent methods for malware detection in Tables 1, 3–7, and 21. The proposed feature selection-based detection technique (LSSVM, RBF with principal component analysis (FR), and BFE (FS)) yielded the best performance in terms of execution time and accuracy.

RQ3: This research question sought to assess the feature selection and examine the link between benign and malicious applications. Various feature reduction techniques were compared to identify the subset of traits capable of detecting whether an app is malicious, as shown in Tables 16–20. The feature subset selected by the BFE (FS) outperforms all other retrieved features in terms of malware detection.

RQ4: The machine learning algorithm performance in terms of feature ranking (FR) is influenced by the features and the type of malware data, as shown in Tables 11 and 12. Six variations of feature ranking algorithms (FR) were used here to find a reduced subset of features. The results of the *t*-test showed that feature selection using principal component analysis (PCA, i.e., the FR6 technique) generated the best results. Tables 11 and 12 show the feature ranking techniques' comparative results for different categories of Android apps in terms of the correlation of the best features and the assessment of subsets obtained by the classifier. The principal component analysis (FR6) outperformed the other feature ranking techniques in terms of accuracy and F-measure.

RQ5: In response to this research question, we compared different approaches to selecting feature subsets in terms of the F-measure and accuracy in the case of supervised, semisupervised, and hybrid machine learning algorithms; and two additional performance parameters, intracluster and intercluster distance, in the case of unsupervised machine learning algorithms, as shown in Tables 16–18. The BFE (FS) recorded a higher performance over the rest of the consideration. The rough set analysis (FS) outperformed the existing feature subset selection techniques, while BFE (FS) outperformed the proposed individual feature subset selection techniques. Overall, BFS (FS) outperformed the existing feature subset selection techniques (FS) and the proposed individual subset selection techniques (FS).

RQ6: Feature subset selection and feature ranking techniques are compared in Tables 11–15. We used the pairwise *t*-test to determine whether the approaches based on feature subset selection performed better than those based on feature ranking, or whether both performed equally well. A significant difference between them was observed, whereby approaches based on feature ranking yielded better results than those based on selected feature subsets.

RQ7: To answer this question, we showed that numerous feature selection (FS) techniques work very well with specific machine learning algorithms. We thus used different feature selection strategies by using a variety of machine learning methods to determine their usefulness. Their performances in terms of accuracy, F-measure, and execution time were recorded. The results showed that the combined BFE (FS) delivered the best performance of all feature selection methods in

terms of all parameters. This BFE (FS) was introduced early in the data engineering stage with principal component analysis (FR), thereby improving LSSVM and RBF (Classifier) overall detection performance, especially when compared with other detection techniques. This is shown in Tables 1, 3–7.

Overall, we compare our proposed technique with the other state-of-the-art methods for Android malware detection in recent years while comparing the efficiency of the proposed technique. To be more precise, we will focus our comparison on the other state-of-the-art feature-based malware detection techniques in recent years. The proposed technique outperformed the other state-of-the-art feature-based malware detection techniques in recent years in terms of accuracy, memory consumption, accuracy, precision, AUC, runtime, and error rate ratio, as shown in Table 21.

We will focus on the discussion under these two headings:

(a). *Performance Characteristics*. To enhance the detection and performance, we proposed a new embedded feature selection-based detection technique (embedded BFE (FS) with LSSVM, RBF, and principal component analysis (FR6)). The proposed detection technique improves the accuracy by 1.5%, the runtime by 12.74(SI), WFM by 6%, AUC by 3%, and FM by 6%, as shown in Table 21. This improvement comes from the introduction of the embedded BFE (FS) feature subset selection algorithm, which helps to select the best features needed for a swift and effective detection process by LSSVM, and RBF (classifier) with principal component analysis (FR6).

(b). *Security Characteristics*. In terms of error rate ratio, runtime, memory consumption, and other factors, the proposed feature selection-based detection technique outperforms other feature selection-based detection techniques. This comes from the introduction of the embedded BFE (FS) feature subset selection algorithm (improves the feature subset selection process), thereby easing the classifier (LSSVM and RBF with principal component analysis (FR6)) by reducing the runtime, memory consumption, improving the error rate ratio, and others.

5. Conclusions

This research focused on techniques for choosing feature subsets by applying a preselected set of criteria to help us determine whether an Android app is malware or benign. The execution involved 30 types of Android applications. Our experiments yielded the following conclusions:

It is possible to determine a limited set of features. The model for malware detection developed using such a collection can accurately detect malware in apps. Our experimental findings showed that considering feature selection helped reduce the size of the feature sets. Models built by using approaches based on feature selection performed better than those built by using all extracted features. The suggested malware detection system is confined to classification and can be extended in future work with additional deep learning approaches.

Data Availability

This dataset was collected from the Mendeley repository and can be assessed by the link: https://data.mendeley.com/datasets/9b45k4hkdf/1.

Conflicts of Interest

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

Acknowledgments

The authors sincerely acknowledge and dedicate this paper to the support of the Ministry of Higher Education, Malaysia, and Universiti Tenaga Nasional (UNITEN), under the Transdisciplinary Research Grant Scheme (TRGS) of Grant no. "TRGS/1/2020/UNITEN/01/1/2." The research and publication of this article were funded by the Ministry of Higher Education, Malaysia, and Universiti Tenaga Nasional (UNITEN), under the Transdisciplinary Research Grant Scheme (TRGS) of Grant no. "TRGS/1/2020/UNITEN/01/1/2."

References

[1] A. Mahindru and A. L. Sangal, "MLDroid: framework for Android malware detection using machine learning techniques," *Neural Computing & Applications*, vol. 33, no. 10, pp. 5183–5240, 2020.

[2] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[3] L. Cai, Y. Li, and Z. Xiong, "JOWMDroid: android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters," *Computers & Security*, vol. 100, Article ID 102086, 2021.

[4] S. Fallah and A. Bidgoly, "Benchmarking machine learning algorithms for malware detection in smartphones," *Jordanian Journal of Computers and Information Technology*, vol. 5, no. 3, p. 1, 2019.

[5] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, Article ID 101663, 2020.

[6] H. Rathore, S. K. Sahay, P. Nikam, and M. Sewak, "Robust android malware detection system against adversarial attacks using Q-learning," *Information Systems Frontiers*, vol. 23, no. 4, pp. 867–882, 2020.

[7] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha, "Two anatomists are better than one—dual-level android malware detection," *Symmetry*, vol. 12, no. 7, p. 1128, 2020.

[8] S. Wang, Z. Chen, Q. Yan et al., "Deep and broad URL feature mining for android malware detection," *Information Sciences*, vol. 513, pp. 600–613, 2020.

[9] A. G. Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: time-based hybrid-featured dataset for effective android

malware detection and characterization," *Computers & Security*, vol. 110, Article ID 102399, 2021.

[10] M. Abuthawabeh and K. Mahmoud, "Enhanced android malware detection and family classification, using conversation-level network traffic features," *The International Arab Journal of Information Technology*, vol. 17, no. 4A, pp. 607–614, 2020.

[11] G. Ditzler, J. LaBarck, J. Ritchie, G. Rosen, and R. Polikar, "Extensions to online feature selection using bagging and boosting," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 9, pp. 4504–4509, 2018.

[12] L. M. Wang and Y. M. Shao, "Crack fault classification for planetary gearbox based on feature selection technique and K-means clustering method," *Chinese Journal of Mechanical Engineering*, vol. 31, no. 1, p. 4, 2018.

[13] D. Lakshmipadmaja and B. Vishnuvardhan, "Classification performance improvement using random subset feature selection algorithm for data mining," *Big Data Research*, vol. 12, pp. 1–12, 2018.

[14] N. Nayar, S. Ahuja, and S. Jain, "Swarm intelligence for feature selection: a review of literature and reflection on future challenges," *Advances In Data And Information Sciences*, vol. 39, pp. 211–221, 2018.

[15] B. Chakraborty and A. Kawamura, "A new penalty-based wrapper fitness function for feature subset selection with evolutionary algorithms," *Journal of Information and Telecommunication*, vol. 2, no. 2, pp. 163–180, 2018.

[16] H. Hichem, M. Elkamel, M. Rafik, M. T. Mesaaoud, and C. Ouahiba, "A new binary grasshopper optimization algorithm for feature selection problem," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 2, pp. 316–328, 2022.

[17] R. Saidi, W. Bouaguel, and N. Essoussi, "Hybrid feature selection method based on the genetic algorithm and Pearson correlation coefficient," *Machine Learning Paradigms: Theory and Application*, vol. 801, pp. 3–24, 2018.

[18] Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An empirical study on the effectiveness of feature selection for cross-project defect prediction," *IEEE Access*, vol. 7, pp. 35710–35718, 2019.

[19] Y. Mao and Y. Yang, "A wrapper feature subset selection method based on randomized search and multilayer structure," *BioMed Research International*, vol. 2019, pp. 1–9, Article ID 9864213, 2019.

[20] A. K. Shukla, P. Singh, and M. Vardhan, "A hybrid framework for optimal feature subset selection," *Journal of Intelligent and Fuzzy Systems*, vol. 36, no. 3, pp. 2247–2259, 2019.

[21] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: effective android malware detection and categorization via app-level profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2019.

[22] H. Cai, "Embracing mobile app evolution via continuous ecosystem mining and characterization," in *Proceedings of the IEEE/ACM 7Th International Conference on Mobile Software Engineering and Systems*, Korea, July 2020.

[23] J. Y. Kim, S. J. Bu, and S. B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep auto encoders," *Information Sciences*, vol. 460-461, pp. 83–102, 2018.

[24] S. M. Sohi, J. P. Seifert, and F. Ganji, "RNNIDS: enhancing network intrusion detection systems through deep learning," *Computers & Security*, vol. 102, Article ID 102151, 2021.

[25] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019.

[26] Z. Moti, S. Hashemi, and A. Namavar, "Discovering future malware variants by generating new malware samples using generative adversarial network," in *Proceedings of the 2019 9Th International Conference on Computer and Knowledge Engineering (ICCKE)*, Iran, October 2019.

[27] S. Sharmeen, S. Huda, J. Abawajy, and M. M. Hassan, "An adaptive framework against android privilege escalation threats using deep learning and semi-supervised approaches," *Applied Soft Computing*, vol. 89, Article ID 106089, 2020.

[28] Y. Wang and J. Zheng, "An evaluation of one-class feature selection and classification for zero-day android malware detection," *Advances in Intelligent Systems and Computing*, pp. 105–111, 2020.

[29] Q. Wen and K. Chow, "CNN based zero-day malware detection using small binary segments," *Forensic Science International: Digital Investigation*, vol. 38, Article ID 301128, 2021.

[30] J. Wu and A. Kanai, "Utilizing obfuscation information in deep learning-based Android malware detection," in *Proceedings of the 2021 IEEE 45Th Annual Computers, Software, and Applications Conference (COMPSAC)*, Spain, July 2021.

[31] S. Millar, N. McLaughlin, J. Martinez Del Rincon, and P. Miller, "Multi-view deep learning for zero-day Android malware detection," *Journal of Information Security and Applications*, vol. 58, Article ID 102718, 2021.

[32] Z. He, T. Miari, H. Makrani, M. Aliasgari, H. Homayoun, and H. Sayadi, "When machine learning meets hardware cyber security: delving into accurate zero-day malware detection," in *Proceedings of the 2021 22nd International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, U.S.A, April 2021.

[33] D. Carlin, A. Cowan, P. O'Kane, and S. Sezer, "The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes," *IEEE Access*, vol. 5, pp. 17742–17752, 2017.

[34] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: self-evolving android malware detection system," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS & P)*, Sweden, June 2019.

[35] H. Cai, X. Fu, and A. L. Hamou, "A study of run-time behavioral evolution of benign versus malicious apps in Android," *Information and Software Technology*, vol. 122, Article ID 106291, 2020.

[36] H. Cai and B. Ryder, "A longitudinal study of application structure and behaviors in Android," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2934–2955, 2021.

[37] A. Mahindru and A. L. Sangal, "FSDroid: - a feature selection technique to detect malware from Android using Machine Learning Techniques," *Multimedia Tools and Applications*, vol. 80, no. 9, pp. 13271–13323, 2021.

[38] A. Mahindru and A. L. Sangal, "SOMDROID: android malware detection by Artificial Neural Network trained using unsupervised learning," *Evolutionary Intelligence*, vol. 15, no. 1, pp. 407–437, 2020.

[39] A. Mahindru and A. L. Sangal, "PARUDroid: validation of android malware detection dataset," *Journal of Cybersecurity and Information Management*, vol. 3, pp. 42–52, 2020.

[40] R. Taheri, M. Shojafar, M. Alazab, and R. Tafazolli, "Fed-IIoT: a robust federated malware detection architecture in industrial IOT," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 8442–8452, 2021.

[41] A. Mahindru and A. L. Sangal, "Hybridroid: an empirical analysis on effective malware detection model developed

using ensemble methods," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 8209–8251, 2021.

[42] Z. Akram, M. Majid, and S. Habib, "A systematic literature review: usage of logistic regression for malware detection," in *Proceedings of the 2021 International Conference on Innovative Computing (ICIC)*, Pakistan, November 2021.

[43] A. H. E. Fiky, "Deep droid: deep learning for android malware detection," *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 12, pp. 122–125, 2020.

[44] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "IoT malicious traffic identification using wrapper-based feature selection mechanisms," *Computers & Security*, vol. 94, Article ID 101863, 2020.

[45] M. Nassar, H. Safa, A. A. Mutawa, A. Helal, and I. Gaba, "Chi squared feature selection over Apache Spark," in *Proceedings of the 23rd International Database Applications & Engineering Symposium*, pp. 1–5, Athens, Greece, June 2019.

[46] A. G. Karegowda, A. S. Manjunath, and M. A. Jayaram, "Comparative study of attribute selection using gain ratio and correlation based feature selection," *International Journal of Information Technology and Knowledge Management*, vol. 2, pp. 271–277, 2010.

[47] H. P. Vinutha and B. Poornima, "An ensemble classifier approach on different feature selection methods for intrusion detection," *Advances in Intelligent Systems and Computing*, vol. 672, pp. 442–451, 2018.

[48] E. O. Omuya, G. O. Onyango, and M. W. Kimwele, "Feature selection for classification using principal component analysis and information gain," *Expert Systems with Applications*, vol. 174, Article ID 114765, 2021.

[49] J. Abawajy, A. Darem, and A. A. Alhashmi, "Feature subset selection for malware detection in smart IoT platforms," *Sensors*, vol. 21, no. 4, p. 1374, 2021.

[50] S. Selvaganapathy, S. Sadasivam, and V. Ravi, "A review on android malware: attacks, countermeasures and challenges ahead," *Journal of Cyber Security and Mobility*, vol. 10, 2021.

[51] M. Nauman, N. Azam, and J. Yao, "A three-way decision making approach to malware analysis using probabilistic rough sets," *Information Sciences*, vol. 374, pp. 193–209, 2016.

[52] N. Şahin, "Malware detection using transformers-based model GPT-2," Master's thesis, Middle East Technical University, Turkey, 2021.

[53] H. M. Rouzbahani, H. Karimipour, A. Rahimnejad, A. Dehghantanha, and G. Srivastava, "Anomaly detection in cyber-physical systems using machine learning," *Handbook of Big Data Privacy*, Springer, Germany, pp. 219–235, 2020.