Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Discovering optimal features using static analysis and a genetic search based method for Android malware detection*

Ahmad FIRDAUS[†‡1,2], Nor Badrul ANUAR[†‡1], Ahmad KARIM[3], Mohd Faizal Ab RAZAK[1,2]

[1]*Department of Computer System and Technology, University of Malaya, Kuala Lumpur 50603, Malaysia*

[2]*Faculty of Computer System & Software Engineering, University Malaysia Pahang, Gambang 26300, Malaysia*

[3]*Department of Information Technology, Bahauddin Zakariya University, Multan 60000, Pakistan*

[†]E-mail: ahmadfirdaus@um.edu.my; badrul@um.edu.my

Received Aug. 22, 2016; Revision accepted Mar. 15, 2017; Crosschecked June 8, 2018

Abstract: Mobile device manufacturers are rapidly producing miscellaneous Android versions worldwide. Simultaneously, cyber criminals are executing malicious actions, such as tracking user activities, stealing personal data, and committing bank fraud. These criminals gain numerous benefits as too many people use Android for their daily routines, including important communications. With this in mind, security practitioners have conducted static and dynamic analyses to identify malware. This study used static analysis because of its overall code coverage, low resource consumption, and rapid processing. However, static analysis requires a minimum number of features to efficiently classify malware. Therefore, we used genetic search (GS), which is a search based on a genetic algorithm (GA), to select the features among 106 strings. To evaluate the best features determined by GS, we used five machine learning classifiers, namely, Naïve Bayes (NB), functional trees (FT), J48, random forest (RF), and multilayer perceptron (MLP). Among these classifiers, FT gave the highest accuracy (95%) and true positive rate (TPR) (96.7%) with the use of only six features.

Key words: Genetic algorithm; Static analysis; Android; Malware; Machine learning
https://doi.org/10.1631/FITEE.1601491 **CLC number:** TP309.5

1 Introduction

Numerous people use mobile devices frequently in their daily activities to accomplish imperative tasks, such as health management, synchronous data transfer, family communications, money transactions, and sensor activities (La Delfa et al., 2016). Among all mobile device operating systems (OSs), Android dominates the smartphone market with a share worth of 366 billion dollars (Thomas, 2015). Furthermore, Android devices are available in a wide range of prices from as low as 50 dollars (eBay, 2016). Given that Android appliances are ubiquitous, Android malware has been rapidly growing in scale. Unscrupulous programmers design malware computer programs to perform diverse malicious actions without users' consent. These actions infiltrate user devices to gain data (e.g., photos, bank activities, messages, phone contacts, and map locations) (Karim et al., 2014). The Sophos mobile security website discovered 610 389 new malware samples outside of the Google Play market in the first six months of 2015 (Komili, 2015). Also, security analysts have discovered hidden malware in 104 applications in the Google Play store; these applications have been downloaded over 3.2 million times in user devices (Russon, 2016). Thus, detecting malware is imperative, as well as countering and reducing its threats and

‡ Corresponding author

* Project supported by the Ministry of Science, Technology and Innovation of Malaysia, under the Grant eScienceFund (No. 01-01-03-SF0914)

ORCID: Ahmad FIRDAUS, http://orcid.org/0000-0002-7116-2643

© Zhejiang University and Springer-Verlag GmbH Germany, part of Springer Nature 2018

risks (Anuar et al., 2013; Khatavakhotan and Ow, 2015; Adewole et al., 2017).

To detect malware in Android applications, security analysts need to conduct an analysis. It is to understand the the content of each application behavior and its code. There are two types of malware analysis, dynamic and static, with some analyses combining these approaches in a hybrid model. Dynamic analysis investigates the behavior of the running processes by executing the applications. For instance, Narudin et al. (2016) observed network system activities to monitor applications and Afifi et al. (2016) used DyHAP to analyze mobile network traffic to expose malware. However, dynamic analysis has its limitations; its coverage is limited because it monitors an application's behavior only within a certain range of time. The behavior of the malware beyond the time of the experiment is not investigated. It triggers only certain activities, so it is possible to miss other actions beyond the analytical range. In contrast, static analysis (Chess and McGraw, 2004) focuses on the malware application code and covers all possible activities without any time restrictions, because the application is unexecuted. The main aim of static analysis procedures is to reverse engineer the applications to retrieve the entire code and then further inspect the structure and substance within the applications (Chang and Hwang, 2007; Sharif et al., 2008; Aafer et al., 2013). Therefore, it could check overall code, and its memory resource requirement is low and the analysis process is fast because the application is unexecuted. In addition, static analysis is able to discover unknown malware with enhanced detection accuracy using machine learning approaches (Feizollah et al., 2013; Narudin et al., 2014). To conduct a proper and accurate detection using static analysis and machine learning, the analysis requires special characteristics or features to differentiate between malware and benign applications.

Features referring to the elements or characteristics are applied to classify an application as either malware or benign. Machine learning classifiers normally use these features as inputs to make a decision. A minimum number of features are desirable because they enhance the accuracy (i.e., a more accurate predictive model) with fewer data, reduce the complexity of the detection model, and decrease the amount of noisy and irrelevant data (Feizollah et al., 2015; Zia et al., 2015; Sarip et al., 2016). Hence, finding a method to select the minimum number of features is necessary. In previous machine learning and static analysis studies (Sarma et al., 2012; Sanz et al., 2013; Talha et al., 2015), the permission category contained the common features. However, some types of malware, such as root exploit, can bypass permission security (Schmidt et al., 2009), and therefore there is a need to investigate and combine special and unique features other than permission features. A number of studies (Arp et al., 2014; Yerima et al., 2014, 2015) combined multiple features from different categories, e.g., application programming interface (API), code-based, strings, and network addresses. The number of categories increases, so does the choice of features. As a consequence, machine learning may face high incremental complexity in detection models for malware classification. Security analysts now face difficulties in selecting the minimum numbers of features within all types of categories. Given that, searching for the minimum number of features from multiple categories is necessary. Therefore, in this study we adopt a method called 'genetic search' (GS), which is based on the genetic algorithm (GA), to search for and improve the parameters to provide a minimum quantity of features from multiple categories.

GA is an algorithm that mimics the natural evolutionary process. It consists of a crossover process that combines multiple generations and continues to loop until the best generations are achieved. In several studies (Punch et al., 1993; Fröhlich et al., 2003; Middlemiss and Dick, 2003; Stein et al., 2005), security analysts have applied GA to achieve the best parameters of features for their detection algorithms to increase accuracy. However, none of these studies adopted the GA method in selecting features for detecting Android malware using static analysis and machine learning. In this study we use GS, which performs as a search based on GA, as described by Goldberg and Holland (1988). GS searches for the best generations of the smallest number of features as possible as it can to detect malware. This approach retrieves GS-selected optimal features in permission, directory path, and code-based categories. For evaluation, we use five machine learning classifiers, Naïve Bayes (NB), multilayer perceptron (MLP), functional trees (FT), J48, and random forest (RF), to test the

efficiency of the features in detecting malware.

In this study, we use static analysis and machine learning to search for the best and smallest number of features to enhance accuracy and reduce the amount of complexity, noise, and irrelevant data in detecting malware. As many end users currently use the Android OS, the most famous mobile device, our experiment focuses on Android.

2 Background

Google introduced Android primarily for mobile device facilities based on the Linux OS (Android Developers, 2015). It is an open-source application, and the merchandise includes smartphones, tablets, smart watches, Android televisions, and Android auto in vehicles. Moreover, Google offers client access to its own services, such as search, YouTube, maps, gmail, photos, calendar, and drives in Android devices. Fig. 1 illustrates the relationships in developing Android appliances, while Fig. 2 introduces the Android OS architecture tiers consisting of applications, application framework, libraries, Android runtime, and Linux kernel. Table 1 describes these Android OS architecture tiers.

3 Related work

We describe previous studies in the malware detection field, including both static and dynamic analyses, feature selection, machine learning, and our proposed method for classifying malware.

3.1 Dynamic and static analysis

Note that the primary goal is to classify malware in Android applications. Security analysts need to perform a malware analysis (Razak et al., 2016). Specifically, the analysis is needed to comprehend the application's behavior and code. Two types of analysis, dynamic and static, have emerged to thwart malware violations. Table 2 lists the advantages and disadvantages of both types.

Dynamic analysis is a technique used to discover malware by performing the application and monitoring its behavior. Narudin et al. (2014) monitored 11 network features consisting of IP address, port number, network frames, and packets. Crowdroid developed by Burguera et al. (2011) examined system call activities. Andromaly (Shabtai et al., 2012) applied central processing unit (CPU), memory, and battery consumption as features. Nonetheless, dynamic analysis (Anuar et al., 2008; Feizollah et al., 2013;

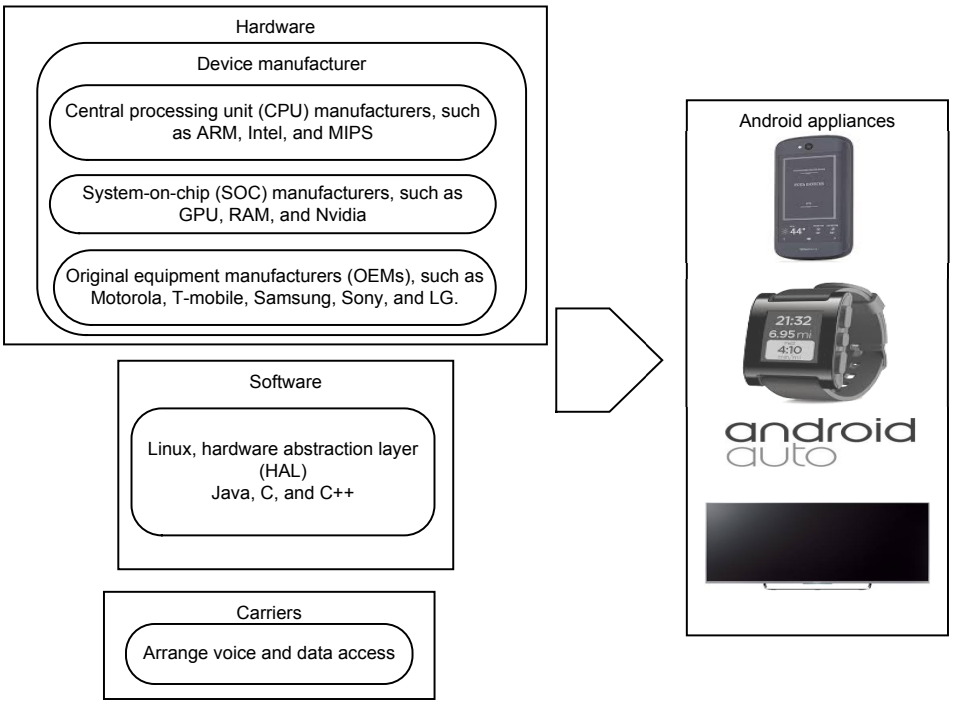

Fig. 1 Android relationship development

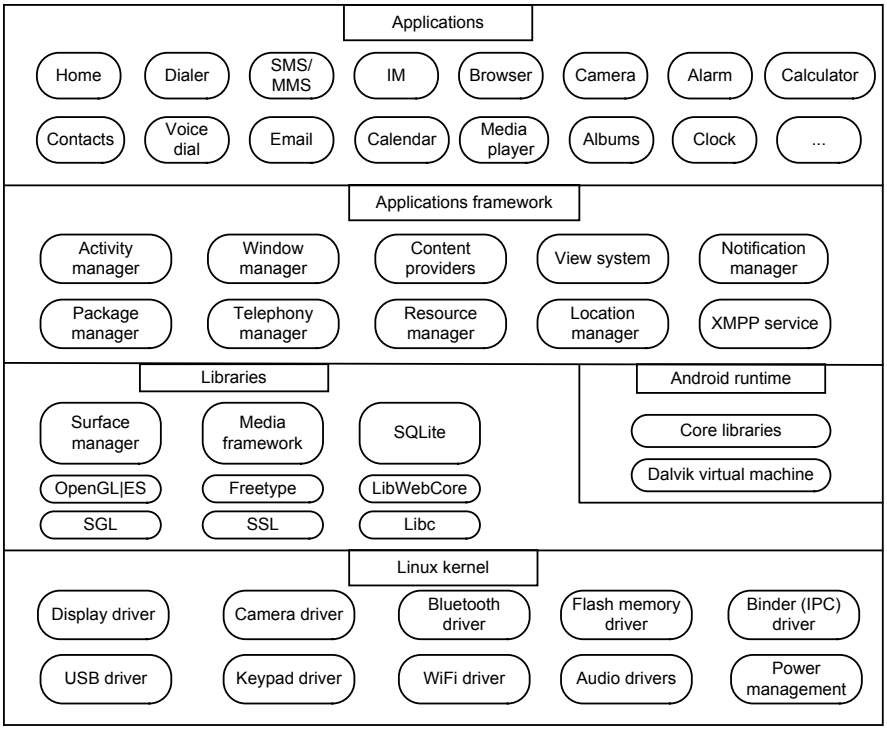

Fig. 2 Android operating system architecture tiers

Table 1 Android operating system architecture tier information

Tier	Information
Applications	This is the top level in the Android architecture. This layer is where the users directly interact.
Applications framework	This layer provides the system server with a process containing the main modules for device management (i.e., content providers, activity, resource, notifications, package, and Window managers). These components interact with the Linux drivers.
Android runtime	This layer comprises the Dalvik virtual machine (DVM), the core component responsible for executing the Dalvik executable format application. DVM is chosen in the Android architecture because of its efficient concurrent execution in a resource-constrained environment. In DVM, Google introduced Android runtime, which offers advanced features, such as ahead-of-time compilation, improved garbage collection, development, debugging enhancement, in the 4.4 version release.
Libraries	This layer consists of a set of C/C++ libraries assigned to invoke basic kernel functionalities.
Linux kernel	This layer, the lowest in the architecture, stores multiple drivers such as wireless fidelity (WiFi), camera, bluetooth, display, universal serial bus (USB), binder (a driver that implements inter-process communication (IPC)), and other necessary drivers. It contains the Linux version built for Android from Linux version 2.6 onwards.

Afifi et al., 2016) needs high specification in both memory and CPU to support the application to run in a certain range of time. Furthermore, this type of analysis may miss malware activities that are beyond the time range of the experiment (attackers trigger malware actions at a certain time whenever they decide).

In contrast to dynamic analysis, static analysis is a procedure for inspecting the code without running the application. The advantages of static analysis are low resource use (e.g., memory and CPU) and fast processing (Chess and McGraw, 2004). The reason for these advantages is that the analysis involves only reverse-engineering the applications and thoroughly scrutinizing the overall code without executing the application. Hence, it covers all possible activities without a time range constraint. Therefore, we focus on static analysis as it could provide a big picture of

the application requiring fewer resources and has a faster analysis process because the application is unexecuted. Nonetheless, Table 2 lists similar limitations of both static and dynamic analyses in selecting a minimum number of features. In detecting malware, features refer to attributes or elements that differentiate an application as either malware or benign. Security practitioners face obstacles in investigating various features in all types of categories (e.g., permission, API, directory path, and code-based) and need to reduce the number of these features. Finding the smallest number of the best features is crucial because it enhances accuracy (i.e., producing a more accurate predictive model) with fewer data and reduced model complexity (Feizollah et al., 2015).

3.2 Static analysis and features

Static analysis primarily reverse engineers an application to retrieve the code for scrutiny. The advantages of this type of analysis are that it is fast and has a low risk of creating a bottleneck situation, because the analysis process examines the application statically, without executing it (Ikinci et al., 2008). In addition, it provides complete code coverage and scalability. Table 3 lists the categories and associates information of features applied in previous static analysis studies for detecting malware.

3.3 Machine learning and optimization of features

Machine learning is a scientific discipline that can predict future decisions and outputs based on the experience gained through past input features (learning set) (Kotsiantis et al., 2006; Feizollah et al., 2013). The learning set is based on a given dataset, and intelligent decisions are made according to certain algorithms. This technique has been widely used for assigning applications to classes (normal or malware). Furthermore, machine learning belongs to the field of artificial intelligence, which allows a computer to reason and make decisions based on datasets (Kotsiantis et al., 2006). Machine learning is divided into two main learning types: unsupervised and supervised.

3.3.1 Unsupervised learning

In unsupervised learning, the unknown application is unlabeled, and the data are assigned to different groups called ‘clusters’. This entails dividing a large dataset into smaller datasets with certain similarities. A given object set is classified through a certain number of clusters (assuming k clusters). The objective is to find k centroids (assigned to each cluster). This unsupervised learning algorithm randomly chooses the centroid from the applications set, takes each application belonging to a given dataset, and assigns it to the nearest centroid.

3.3.2 Supervised learning

In supervised learning, each application in the dataset for training should be labeled with the category. The label is the class of each application (malware or benign). Each training dataset contains an input (feature or characteristic) and an output (class label-malware or benign). Afterwards, the approximate distance between the input and output examples is calculated in the training to create a model. The model should be capable of classifying unknown applications as malware or benign applications. Our

Table 2 Advantages and disadvantages of dynamic and static analysis

Type of analysis	Advantages	Limitations
Dynamic	Unknown malware detection ability; detection of normal applications that change to malware on-the-fly	Possible miss of malware activities beyond the analysis range; difficulty in detecting applications which are able to hide malicious behavior while runing; difficulty in finding minimal features (e.g., traffic, memory) to detect malware
Static	Low resource uses (e.g., CPU, memory, network, and storage); relevance in mobile devices equipped with low specifications; fast processing in reverse engineering the application and examining the code	Inability to detect unknown malware; inability to detect normal applications that change to malware on-the-fly; difficulty in finding minimal features (e.g., permission, function call, and strings) to detect malware

Table 3 Series of features used in previous static analysis studies

Feature	Information	Reference(s)
API	Containing codes of an application consisting of classes, methods, functions, and parameters	Aafer et al., 2013; Lee and Jin, 2013; Deshotels et al., 2014
Function call	In application code, it is a declaration in an argument. It either contains any number of names separated by commas or is empty	Schmidt et al., 2009; Gascon et al., 2013
Code structures	Comprising a line or set of programming codes in an application	Suarez-Tangil et al., 2014
Sources and sinks	Sources and sinks are related terms. The sources in computing areas are where the data enter the program, whereas sinks are where the data flows out of the program (Rasthofer et al., 2014)	Lu et al., 2012; Gordon et al., 2015
Bytes	Referring to codes in an application	Faruki et al., 2013
String	A feature indicating a plain-text string or a sequence of characters such as ‘root’, ‘exec’, and ‘password’. Malware includes certain strings to execute malicious activities	Firdaus and Anuar, 2015
Reverse-engineered life cycle model	Android applications consist of essential building blocks called ‘application components’ (activity, service, broadcast receiver, and content provider) which follow a life cycle model during execution	Junaid et al., 2016
AndroidManifest.xml	One of the files in Android applications is AndroidManifest.xml. It is an essential file, containing the package name, the application components (activities, services, broadcast receivers, and content providers), the permission declarations, the instrumentation classes, the API minimum level, and the list of necessary libraries (Android, 2015)	Peng et al., 2012; Sahs and Khan, 2012; Sarma et al., 2012; Walenstein et al., 2012; Wu et al., 2012; Aung and Zaw, 2013; Huang et al., 2013; Samra et al., 2013; Sanz et al., 2013; Talha et al., 2015
Intent	Intent objects deliver an abstract definition of the operations in an application which it plans to accomplish	Feizollah et al., 2017
API and permission	In the Android OS process, permission and API are dependent on each other. Parts of API calls in the code need permission to execute (Wu et al., 2012)	Bartel et al., 2012; Grace et al., 2012; Wu et al., 2012; Aafer et al., 2013; Liang et al., 2013; Peiravian and Zhu, 2013; Zhou et al., 2013; Arp et al., 2014; Arzt et al., 2014; Feng et al., 2014; Huang et al., 2014; Yerima et al., 2014; Sheen et al., 2015
API, permission, and others	Combined features consist of API, permission, and others	Apvrille and Strazzere, 2012; Zhang et al., 2013; Yerima et al., 2014
API and other features (excluding permission)	Combined features consist of API and others, except permission	Seo et al., 2014
Permission and other features	Apart from API combinations, security analysts also combine permission with other features	Shabtai et al., 2010; Grace et al., 2012c; Sarma et al., 2012; Yang and Yang, 2012; Zhou et al., 2012; Yerima et al., 2013, 2014; Kang et al., 2015
Other features except for API and permission	Features used other than API and permission	Grace et al., 2012; Lee et al., 2015

study falls into the supervised learning technique category because it evaluates features of each application along with labels (i.e., malware and benign). To compare results for various machine learning classifiers in different categories, we choose five classifiers. Fig. 3 shows the three types of machine learning classifier categories applied in our experiment, and Table 4 lists the associated information.

Table 5 lists the machine learning classifiers in previous static analysis studies for Android and demonstrates that security analysts have used NB and RF more than the other classifiers (i.e., J48, MLP, and FT). Furthermore, they used MLP only once, and FT was excluded from their analyses. The reason we choose NB and RF is that security analysts have regularly used these classifiers and with great success in the intrusion detection system area (Díaz-Uriarte and de Andrés, 2006; Caruana et al., 2008; Yu et al., 2013). We choose J48, MLP, and FT because previous studies infrequently used these three classifiers as a part of static analysis investigations. Hence, in this study we aim to compare both regularly and infrequently used machine learning classifiers. Moreover, we select these types of classifiers to compare the results for different categories. MLP represents the function, NB the Bayes, and FT, J48, RF the trees categories.

In the interest to achieve good results in machine learning, selecting a minimum number of features is

Table 4 Machine learning information

Category	Machine learning classifier	Advantages
Bayes	NB	1. It operates on the (naïve) assumption of independence of all the features. For example, a fruit may be considered an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other, they independently contribute to the probability that the fruit is an apple, which is why it is known as 'naïve'. 2. It performs quite well in certain real-life applications, such as file classification and spam filtering. 3. It is capable of learning and classifying categories in an extremely rapid manner.
Decision tree	FT, J48, and RF	1. A model is constructed by taking an instance following the path of satisfied conditions starting with the root of the tree until reaching the leaf, which will relate to its corresponding class label. This decision is then converted to detection model rules. 2. It is generally known as 'divide and conquer' algorithms (Kotsiantis, 2013).
Functions	MLP	1. It is one of the function categories of machine learning classifiers applied in many fields because of its stable learning algorithm (Lippmann, 1987). 2. It is an artificial neural network based on biological neural networks, and consists of three layers (input, hidden, and output).

Table 5 Previous static analysis studies

Machine learning classifiers in previous studies	Machine learning classifiers in our study					Reference
	NB	FT	J48	RF	MLP	
Decision tree, NB, Bayesian network, part, boosted Bayesian network, boosted decision tree, RF, and voting feature interval (VFI)	√			√		Shabtai et al., 2010
Simple logistic, NB, Bayesian network, sequential minimal optimization, instance-based learning with parameter *k*, J48, random tree, and RF	√		√	√		Sanz et al., 2013
NB, part, Ridor, decision tree, and simple logistic	√					Yerima et al., 2014
SVM, J48, bagging, prism, and *K*-nearest neighbor			√			Peiravian and Zhu, 2013
RF, random tree, NB, decision tree, and simple logistic	√			√		Yerima et al., 2015
NB, support vector machine (SVM) with sequential minimal optimization (SMO), radial basis function (RBF) network, MLP, Liblinear, decision tree, and RF	√			√	√	Chan and Song, 2015

vital. Table 6 tabulates information about the methods used for selecting features. It provides information from previous works and our study. A study in Drebin (Arp et al., 2014) analyzed the joint vector space and identified the typical patterns of the features in geometric form. Karim et al. (2016) used the element tree xml API, including regular expression, to identify the string or number of attributes to obtain the features.

The forward greedy algorithm (Zhang, 2009) has also been used for feature selection. This algorithm is part of an investigation of sparse approximation (Tropp, 2004). The fundamental aspect of a forward greedy algorithm is that it greedily selects another feature at every iteration and may decrease the loss. Using this strategy, it can generally locate the best features. Forward greedy algorithms follow the problem—locally solving heuristic of making the optimal decision in every phase while searching for the global optimum. This analysis, generalized to the case of measurement noise, is able to identify features in a sparse eigenvalue condition. This capability continues as long as each nonzero coefficient is larger than the constant time noise level. However, forward algorithms cannot correct previous mistakes. Hence, Lai et al. (2011) combined the forward method with the backward greedy algorithm to resolve this downside. As the inverse of forward, a backward greedy algorithm removes each feature in each iteration. In contrast to the joint vector space, element tree, and forward and backward greedy algorithms, we apply an evolutionary algorithm (i.e., population, generation, crossover, and mutation) to search for optimal and relevant features in multiple categories. We then adopt GS, based on a bio-inspired GA, to select a minimum number of features in multiple categories to detect malware using the five machine learning classifiers (i.e., NB, FT, J48, RF, and MLP).

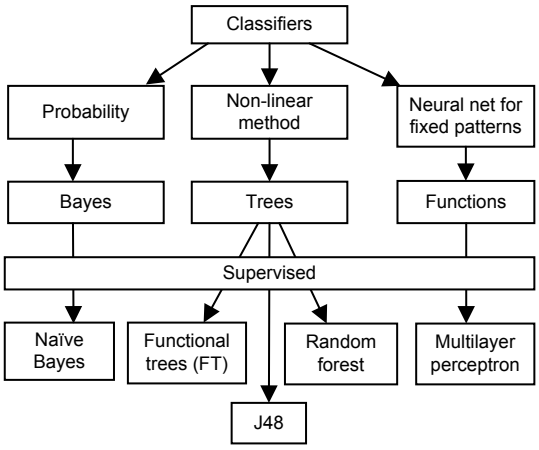

Fig. 3 Machine learning categories

3.4 Genetic algorithm and genetic search

In detecting malware, feature selection has a significant effect on experimental results. We apply GS to select the best and smallest number of features. GS is inspired by evolutionary biology and provides a technique to automatically improve parameters or features. Table 7 lists related works where GA was employed to select features. GA has been widely used in numerous fields, such as soil classification, colon cancer identification, gene expression, and malware intrusion detection.

Punch et al. (1993) used feature selection and data classification for soil classification using GA combined with *K*-nearest neighbor (KNN). KNN enables the measurement of similarity between samples. However, it cannot determine the relative importance of features to discriminate known samples. Hence, GA and KNN are used as the main part in the evaluation stage to optimize the features and classification. This helps to optimize classification through the search for optimal features. The dataset for their study is rhizosphere data, which consists of natural set. This dataset represents part of the soil ecosystem where plant roots, soil, and soil biota communicate

Table 6 Methods for selecting features

Method	Information	Reference
Joint vector space	Used to identify patterns of features	Arp et al., 2014
Element tree xml API and regular expression	Used to wrap an element structure or string	Karim et al., 2016
Forward greedy algorithm	Greedily selecting another feature in each iteration	Zhang, 2009
Forward-backward greedy algorithm	Greedily selecting and removing another feature in each iteration	Lai et al., 2011
Genetic search (GS)	Adopting an evolutionary algorithm to select features	Our study

Table 7 Previous studies on selecting features using GA

No.	Objectives	Reference
1	Classifying soil to three environments: near the roots of a crop (rhizosphere); away from the influence of the crop roots (non-rhizosphere); from a fallow field (crop residue)	Punch et al., 1993
2	Classifying colon cancer and gene expression	Fröhlich et al., 2003
3	Detecting malware intrusion (dynamic analysis)	Middlemiss and Dick, 2003
4	Detecting malware intrusion (dynamic analysis)	Stein et al., 2005
5	Investigating and identifying the strings and applying GS to select the best features for classification (static analysis)	Our study

with each other. These connections offer an advantage to plant, enhance soil fertility, and promote the breakdown of harmful chemicals. They attempted to classify three rhizosphere classes: rhizosphere, non-rhizosphere, and crop residue. In contrast, we apply GS based on GA to detect malware, specifically on the Android OS, and aim to distinguish only two classes: malware and benign.

Fröhlich et al. (2003) applied GA to classify three classes (i.e., toy data, colon cancer, and gene in a yeast dataset). The genetic evolutionary process in the GA enabled it to switch different factors and optimize parameters in a support vector machine (SVM) algorithm. They also used theoretical bounds on the generalization error for SVM and proposed a decimal encoding, which is much more efficient than binary encoding. If the number of features to be selected is unfixed beforehand, the usual binary encoding is preferred. Furthermore, kernel parameters such as the regularization parameter C of SVM can be optimized by GA for the selection of a feature subset, given that the choice of the feature subset influences the appropriate kernel parameters, and vice versa. In this study, we apply GA to select the optimal features among 106 features and evaluate the approach applied to three different categories of machine learning: function (MLP), Bayes (NB), and trees (FT, J48, and RF).

Middlemiss and Dick (2003) used GA to select the optimal features with weighted feature extraction and machine learning in a malware detection study using dynamic analysis. To accomplish this, they implemented GA to calculate the weights for the dataset features. Thereafter, a KNN classifier was applied to use the fitness function through GA and assess the performance of the new weighted feature set. Their results showed that the weighted set of features for the class classification of data can provide an incremental gain in intrusion detection accuracy. In contrast, we apply the static approach instead of dynamic analysis. While their study focused on a computer-based intrusion detection system, our study was specifically concerned with detecting malware in Android mobile devices.

Stein et al. (2005) also applied GA to detect malware. They used the GA algorithm as a method to select a subset of features to be an input in decision tree classifiers using static analysis. They used knowledge discovery and datamining tools competition (KDDCUP) 99 as a dataset to train and test the tree classifier. Their study differed from ours in the detection area: they concentrated on the server, while we specifically targeted malware in the Android platform. We use GA to select the best and the minimum number of features in four categories (e.g., permission, code-based, directory path, and system command) and evaluate those features in five machine learning classifiers.

4 Methodology

Fig. 4 illustrates the four phases of the experimental process. The first phase is data collection, which includes a reverse engineering process during which the code of the application is retrieved. The second phase is string identification, which consists of permission, the words in the double quotes, function, intent, Linux command, directory path, and system command. The third phase applies the GS process to select the best features among all the extracted strings obtained in the string identification phase. The fourth phase involves the machine

learning classifier. The machine learning classifier trains the information in the dataset to construct a detection model which can predict an application to be either benign or malware.

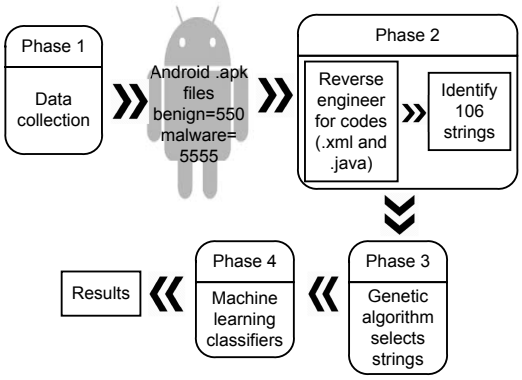

Fig. 4 Structure of the experimental phases

4.1 Data collection

Initially, for the benign dataset, the total number of downloaded applications from the Google Play store (Google, 2014) was 1209. To completely validate the dataset as benign, VirusTotal (https://www.VirusTotal.com/) screened these files by scanning. This process was accomplished by using more than 40 online antivirus applications. During the process of scanning all the 1209 benign applications, certain antivirus applications considered some samples as nearly harmful. For instance, if a scanning result of 1/50 was received, it means that one in 50 antivirus programs considered the sample as malware, whereas the remainder considered it to be benign. Therefore, the benign samples included in our experiment were those that received 0/50 scores, indicating that all antivirus considered them as benign. The total number of applications with a 0/50 score amounted to only 550 samples. Once the benign samples were identified, the subsequent process was concerned with the malware dataset.

On the malware side, this experiment used Drebin (Arp et al., 2014). The entire Drebin dataset consists of 5560 samples in 179 different malware families. However, errors were encountered in five samples during the reverse engineering process; therefore, the final total number of samples in the malware dataset was 5555. Table 8 summarizes the datasets used in the experiment.

Table 8 Dataset summary

Dataset	Source	Number of downloaded samples	Number of used samples
Malware	Drebin	5560	5555
Benign	Google Play store	1209	550

To gain the application code of the dataset, the Jadx (Skylot, 2015) tool was used to reverse-engineer the Android .apk files. Jadx can reverse the .dex file in the .apk Android file to a .java extension file. Once the process was accomplished, we obtained all the files in the nested folders that consisted of .xml and .java file extensions. The common file found after the reverse process was Androidmanifest.xml, and the common folder was called ‘res’. The Androidmanifest.xml file contained the Android permission and intent filters including libraries, and the folder ‘res’ contained the application layout with various .xml file types. The remainder of the folders contained .java file types, in which the file names differed depending on the application.

4.2 String identification

After obtaining the code of the application, the next step is to investigate and identify the strings for features. Fig. 5 illustrates the second phase for string identification. To differentiate the colons, quotes, and brackets in the code, the natural language toolkit (NLTK) (Bird et al., 2009) tokenized the line and pulled out the strings. Afterwards, the existences of each feature (where 1 indicated existence and 0 indicated non-existence) in both benign and malware were determined. In some cases, strings were confused with one another. For example, cat, one of the Linux commands, was confused with other words such as concatenate, locate, and other similar words which contain cat strings. To observe the entire exact line of codes and pull out the cat command, we used the Grep command in Ubuntu terminal.

Other than the Linux commands, the features listed in Sections 4.2.1−4.2.4 were selected from other studies and additional investigation in this experiment. The permission features consist of 13 dangerous permissions declared by the Androguard (Desnos, 2015) open-source tool. The additional features were the proposed set in a root exploit study

(Firdaus and Anuar, 2015) and consisted of the system command, directory path, and code-based features. This set was included because we attempted to detect all malware types, including root exploit. Root exploit is among the malware types that exploit the vulnerable Android kernel to gain root and execute malicious actions, such as installing botnets, spreading Trojans, providing fake antivirus results, and executing root privileges.

4.2.1 Code-based features

Table 9 lists the 36 code-based features along with their existence in each class (malware or benign). For instance, createSubprocess is a code string existing in 307 malware samples, however found none in benign samples. As the malware dataset contained 5555 samples, and the benign dataset only 550, for each dataset it was necessary to calculate the percentage by dividing the existence of the features by the total, and multiplying the result by 100. Fig. 6 shows the existence in the code-based features in percentage form. This shows that benign existence is more than existence of malware features in the code-based category.

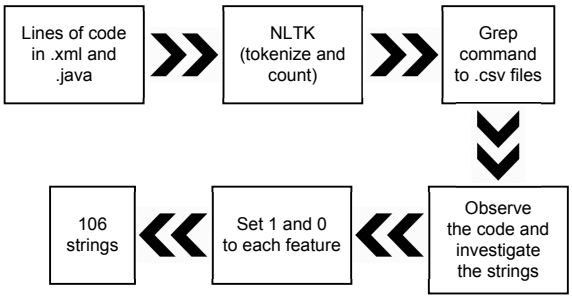

Fig. 5 String identification

4.2.2 Permission features

The second category is the permission features included in the AndroidManifest.xml file. This file is required in every Android application main directory. It represents the essential information regarding the application package name, permission, activities, services, broadcast receivers, and content providers. Table 10 lists the permissions before the GS feature selection phase. For example, Android.intent.action.MAIN is a permission that occurred 5355 times in the malware and 541 times in the benign samples. Some of the permissions taken from the Androguard (Desnos, 2015) tool were declared dangerous, whereas others were added from our investigation. Fig. 7 depicts the existence of these features in percentage form. The plot shows that malware and benign classes

Table 9 Code-based features

No.	Code-based	Number of occurrences		No.	Code-based	Number of occurrences	
		Malware	Benign			Malware	Benign
1	Android.util.Log	4607	543	19	get(os.version)	531	5
2	Android.os.Handler	3878	539	20	getDefaultHost()	489	38
3	DefaultHttpClient	3413	469	21	createSubprocess	307	0
4	setReadTimeout	2719	406	22	com.google.update.RU.U11	272	0
5	UrlEncodedFormEntity	2589	311	23	Forked	95	1
6	getResourceAsStream	2386	261	24	ImageTestActivity	74	0
7	getNetworkInfo	2157	348	25	InstallApk	73	1
8	vnd.Android.package-archive	2034	90	26	ACTION_CostInfo	72	0
9	getExternalStorageState	1954	438	27	ACTION_SaveID	72	0
10	checkPermission	1893	428	28	AndroidThemeService	72	0
11	Cipher.getInstance	1738	408	29	CostNumberConfirm	72	0
12	Math.random	1590	346	30	CostParms	72	0
13	getLaunchIntentForPackage	1413	284	31	getResponseContentGet	72	0
14	getSimSerialNumber	1363	36	32	getResponseContentStream	72	0
15	setWifiEnabled	772	46	33	KeyCostTips	72	0
16	process.waitFor	738	4	34	KeyPIDIntent	72	0
17	getField(manufacturer)	641	29	35	TASK_network_setting	72	0
18	getWifiState	559	25	36	config.dat	72	3

Table 10 Permission features

No.	Permission	Number of occurrences		No.	Permission	Number of occurrences	
		Malware	Benign			Malware	Benign
1	Android.intent.action.MAIN	5355	541	22	Android.intent.extra.shortcut.INTENT	1352	66
2	Android.content.Context	5320	548	23	Android.intent.extra.shortcut.NAME	1352	66
3	Android.permission.INTERNET	5234	527	24	Android.permission.READ_CONTACTS	1314	59
4	Android.intent.category.LAUNCHER	5150	530	25	Android.permission.WRITE_SMS	1213	6
5	Android.telephony.TelephonyManager	4902	481	26	com.Android.launcher.action.INSTALL_SHORTCUT	1197	52
6	Android.intent.action.VIEW	4844	544	27	Android.permission.CHANGE_WIFI_STATE	983	85
7	Android.permission.READ_PHONE_STATE	4838	388	28	Android.intent.action.SCREEN_ON	912	107
8	Android.permission.WRITE_EXTERNAL_STORAGE	3644	447	29	Android.intent.extra.shortcut.ICON_RESOURCE	762	55
9	Android.intent.action.BOOT_COMPLETED	3539	143	30	Android.intent.action.SIG_STR	671	1
10	Android.webkit.WebView	3453	494	31	Android.intent.action.BATTERY_CHANGED_ACTION	581	0
11	Android.content.IntentFilter	3077	504	32	com.google.update.UpdateService	406	0
12	Android.permission.SEND_SMS	2976	72	33	com.google.update.Receiver	398	0
13	Android.webkit.WebSettings	2425	396	34	com.Android.packageinstaller	335	5
14	Android.permission.RECEIVE_SMS	2127	17	35	Android.permission.READ_EXTERNAL_STORAGE	323	82
15	Android.permission.ACCESS_COARSE_LOCATION	2119	297	36	com.google.map.apk	275	0
16	Android.permission.ACCESS_FINE_LOCATION	2109	306	37	Android.intent.action.NEW_OUTGOING_CALL	241	3
17	Android.permission.WAKE_LOCK	2098	334	38	Android.intent.extra.PHONE_NUMBER	173	3
18	Android.permission.READ_SMS	2037	12	39	Android.provider.Telephony.WAP_PUSH_RECEIVED	173	2
19	Android.intent.action.DIAL	1729	184	40	Android.settings.WIRELESS_SETTINGS	158	46
20	Android.provider.Telephony.SMS_RECEIVED	1446	24	41	Android.provider.Telephony.MMS_RECEIVED	72	0
21	Android.intent.action.SCREEN_OFF	1407	286	42	com.Android.browser.application_id	45	28

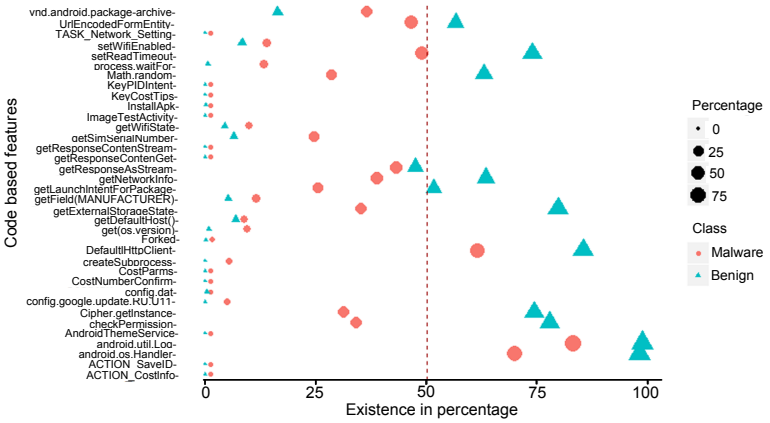

Fig. 6 Existence of code-based features expressed as percentages

The vertical dashed line in the middle distinguishes the benign class with only three malware features included within the range from 50% to 100%

are situated in a similar area from 0% to 100%, which shows no significant difference.

4.2.3 Directory path features

Table 11 lists the directory path features. Some paths were taken from the root exploit study (Firdaus and Anuar, 2015) and others were added through additional investigation. One of the directories, system/bin/su, occurred 633 times in malware but was not found in benign samples. It stores the super user information of the Android OS and attracts unscrupulous authors wishing to gain root in the victim's mobile devices. Fig. 8 shows the percentage of the directory path features in the malware and benign datasets. The figure identifies the critical directory system/bin/secbin, which appears in 44% of the malware samples, a far greater percentage than that for any other feature.

4.2.4 System command features

The final category is composed of the system command features as listed in Table 12. As the objective is to detect all types of malware including root exploit, we include the system command from the root exploit study (Firdaus and Anuar, 2015). These features originated from two categories: Unix commands and Android debug bridge (ADB) commands. Examples of Unix commands are chmod, chown,

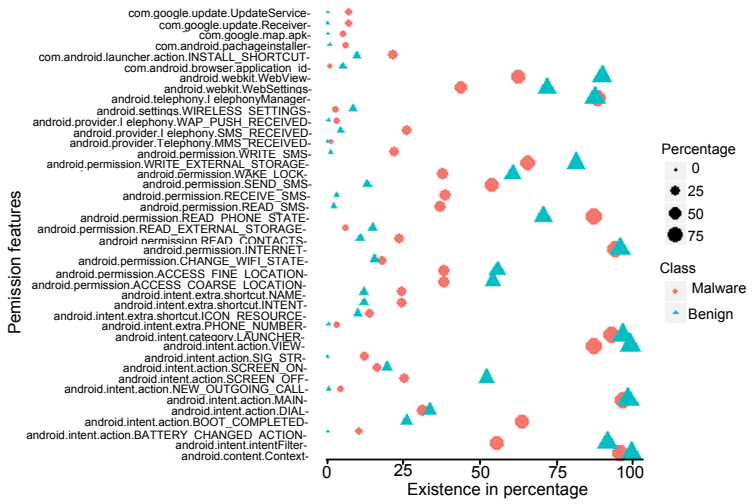

Fig. 7 Existence of permission features expressed as percentages

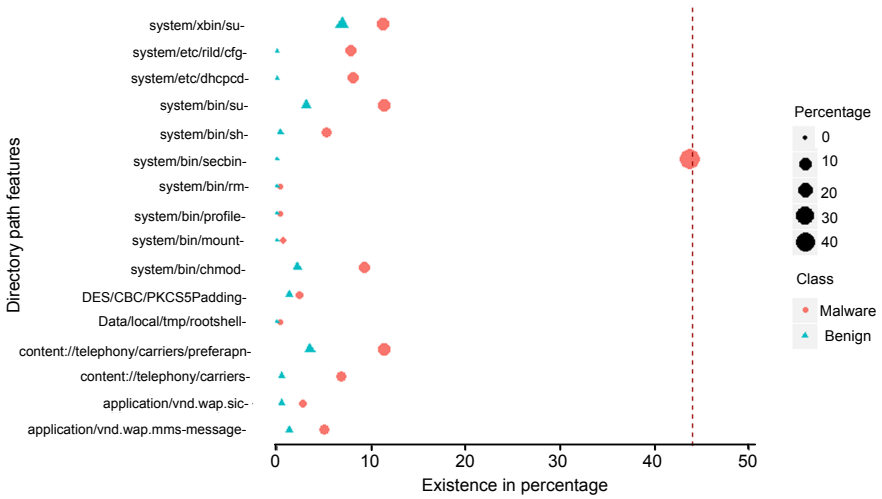

Fig. 8 Existence of directory path features expressed as percentages

Table 11 Directory path features

No.	Directory path	Number of occurrences	
		Malware	Benign
1	system/bin/secbin	2434	0
2	system/bin/su	633	17
3	content://telephony/carriers/preferapn	625	19
4	system/xbin/su	621	38
5	system/bin/chmod	516	12
6	system/etc/dhcpcd	449	0
7	system/etc/rild/cfg	441	0
8	content://telephony/carriers	376	3
9	system/bin/sh	287	2
10	application/vnd.wap.mms-message	280	7
11	application/vnd.wap.sic	151	3
12	DES/CBC/PKCS5Padding	134	7
13	system/bin/mount	41	0
14	data/local/tmp/rootshell	24	0
15	system/bin/rm	23	0
16	system/bin/profile	19	0

pm install, cat, cp-rp, and mount-o remount, and the ADB commands include startservice-n. Fig. 9 displays the existence of these commands in percentage form. The commit command was the most prevalent in both benign and malware classes.

Fig. 10 shows the regression line describing the relationships between significant features in the malware and benign samples. The three rising lines (indicated as code-based, permission, and system command) indicate a positive relationship; the percentage of benign features increases, so does the percentage of malware features. Fig. 10 shows that the malware features used are suitable attributes for detecting malware. However, the line of the directory path category is minimal, because the percentages are so low compared to those of the other categories.

4.3 Proposed genetic search and features

GS is based on the GA method, inspired by and based on the evolutionary biology of nature genetic element (crossover and mutation). This evolutionary process provides a technique to automatically

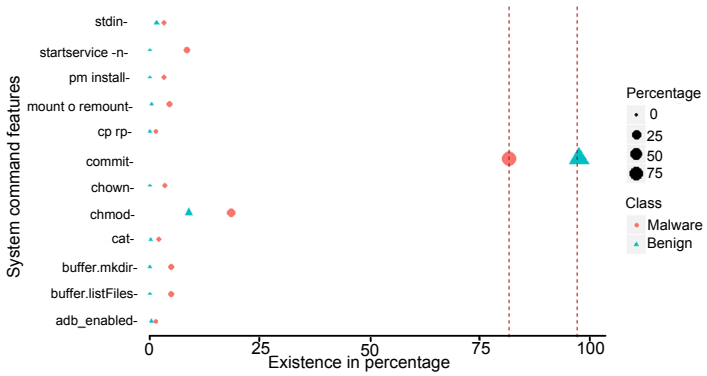

Fig. 9 Existence of system command features expressed as percentages

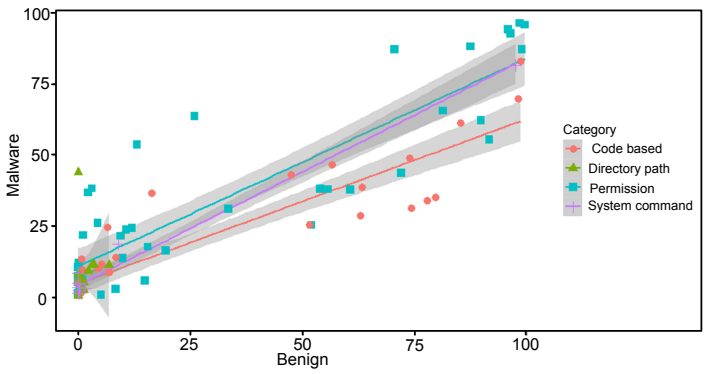

Fig. 10 Regression lines of all categories

improve characteristics or features to generate the best generation. This evolution can transform the worst generation into the best, according to fitness requirements. This method includes a set of requirements. When one generation is unmatched according to certain characteristics, GA excludes that generation, and this process repeats until the best generation is achieved. Furthermore, if GA cannot solve any research problem by providing a final ultimate solution, it will at least serve as a method to select the optimal features. Fig. 11 illustrates the basic GA processes of crossover and mutation.

Table 12 System command features

No.	System command	Number of occurrences	
		Malware	Benign
1	commit	4529	536
2	chmod	1034	49
3	startservice-n	474	0
4	buffer.listFiles	272	0
5	buffer.mkdir	272	0
6	mount-o remount	247	2
7	chown	195	0
8	pm install	183	0
9	stdin	179	8
10	cat	123	1
11	adb_enabled	80	2
12	cp-rp	72	0

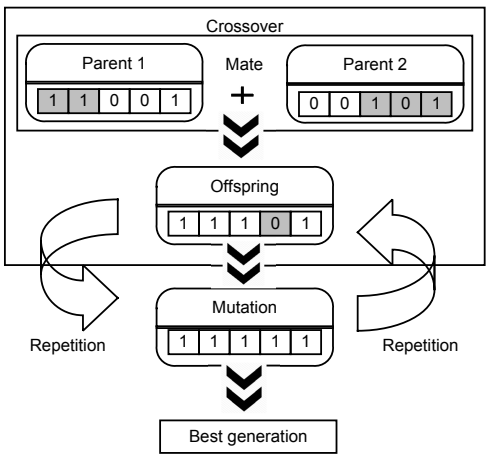

Fig. 11 Basic genetic algorithm process

To further illustrate GA, consider a company requiring advice in designing a good vehicle characteristic and enabling a vehicle to enter and successfully exit from a high hill forest full of obstacles. In this case, the fitness is set according to multiple checkpoints in the forest. First, the vehicle consists of small tires with a low-performance engine. The vehicle keeps going and changing its characteristics or features simultaneously. Once the vehicle successfully reaches the first checkpoint, the first fitness is satisfied and the characteristics of the vehicle are saved. The GA process continues (the characteristics of the vehicle are gradually changing) until the vehicle passes all checkpoints and all the required characteristics are saved. Finally, the set of features of a vehicle necessary to enter and survive the forest is successfully achieved, including big tires, high acceleration engine, big tank, and good quality brakes. Eq. (1) is defined according to the F-function and feature string of length 1 known as the fitness. New features are created from the current population and the probability that a parent string H_j (j=1, 2, …, N) is selected from the N strings:

$$p(H_j)=\frac{f(H_j)}{\sum_{n=1}^{N} f(H_n)}. \tag{1}$$

In selecting features, it is imperative to achieve the minimum possible number of features that is necessary for developing an effective malware detection system. This is crucial as it removes noise and irrelevant data from the dataset (Feizollah et al., 2015) and leads to more accurate machine learning algorithm results. Hence, the next process involves selecting the best features among the 106 features. In the GS process the Waikato environment knowledge analysis (WEKA) tool (Hall et al., 2009) is used. As the total dataset consists of 6105 applications (both benign and malware), the populations size was set as 400 in each generation, giving crosses amounting to a total of 15 generations. The crossover probability was set as 0.6 to produce the offspring list of 106 features. To maintain the value of the feature (0 or 1) in each application, we set zero for the mutation process. At the end of the process, GA successfully identified the best generations of features and selected the six features as listed in Table 13.

These features represent three types of categories: Android permission (read, receive, and write short message service (SMS)), code-based

(checkPermission and com.Android.browser.application.id), and directory path (system/bin/secbin). Table 13 shows that malware existence is more than that of benign features in permission and directory path. In contrast, in the code-based category, the malware existence was less than that of benign. This demonstrates that the GS search for the optimal features is not made according to malware existence, but merely to the evolutionary process in GS.

Table 13 Features selected by genetic search

No.	Feature	Existence in malware (%)	Existence in benign (%)
1	Android.permission.READ_SMS	36.7	2.2
2	Android.permission.RECEIVE_SMS	38.3	3.1
3	Android.permission.WRITE_SMS	21.8	1.1
4	checkPermission	34.1	77.8
5	system/bin/secbin	43.8	0
6	com.Android.browser.application_id	0.8	5.1

4.4 Machine learning classification

In constructing the machine learning model, the classifiers were run using WEKA (Frank et al., 2016). For this tool, it is necessary to prepare a comma separated values (.csv) file. As GS selected the six features in addition to the class label, this file contained seven columns. The file contained 6105 rows, representing the combination of the malware (5555) and benign (550) samples. Given that in this work we used static analysis, each row was composed of 1 or 0. Each row represented an application showing 1 (if the feature existed) or 0 (if the feature was non-existent).

Once the .csv file application was complete, the next step was to convert the file to attribute-relation file format (.arff). This conversion was done by using the WEKA tool. ARFF is an ASCII text file format, in which WEKA specifically introduced to enable faster loading (Williams, 2010). To achieve natural and acceptable results, the evaluation process applied the randomize option to randomly shuffle the order of both classes (malware and benign) in the datasets. Therefore, the class categories in each application were randomly arranged to provide a natural order. Fig. 12 displays the captured screen of some parts in the ARFF file after applying the randomize option.

```
@relation genetic

@attribute android.permission.READ_SMS numeric
@attribute android.permission.RECEIVE_SMS numeric
@attribute android.permission.WRITE_SMS numeric
@attribute checkPermission numeric
@attribute system/bin/secbin numeric
@attribute com.android.browser.application.id numeric
@attribute Class {B,M}

@data
0,0,0,1,1,0,M
0,0,0,1,0,0,B
1,1,1,1,1,0,M
1,0,1,0,1,0,M
0,0,0,0,0,0,B
```

Fig. 12 Part of the attribute-relation file format file

In constructing the machine learning predictive model, it is necessary to use k-fold cross validation methods, which repeatedly run k times. In the experiment, the kth subset served as the test set, and the remaining $k-1$ subsets as the training set. The average of all k trials was computed for the evaluation (Schneider, 2016). We used the 10-fold method, which repeatedly ran 10 times to evaluate feature effectiveness. The dataset was randomly divided into 10 subsets of equal size and the evaluation was repeated 10 times. In each repetition, one subset was used as the test set and the other nine subsets were combined to form the training set. Accordingly, the test set was excluded from the training set, which was used to detect unknown malware in this step.

5 Evaluation and results

The evaluation calculates the measures according to the confusion metrics applied to machine learning classifier evaluation, and the results are presented below.

5.1 Evaluation

To evaluate the effectiveness of the six features selected by GS, it is crucial to address the machine learning classifier performance matrix. The performance measures include accuracy, true positive rate (TPR), false positive rate (FPR), receiver operating characteristics (ROC), precision, recall, and F-measure, the F-measure (F_1 score or F score) is a

measure to test the accuracy and is defined as the weighted harmonic mean of the precision and recall of the test. Table 14 lists the information on the experimental evaluations followed by the corresponding measurement equations which are used in the following sections.

5.2 Results in cross validation

Table 15 lists the results derived from the experiments. In cross validation, FT achieved magnificent results across four categories: highest accuracy (94.22%), highest number of correctly classified instances (5752), lowest number of incorrectly classified instances (353), and lowest FPR (23.8%). In terms of the true positive value, all classifiers gained the same mark except for J48, which achieved the lowest mark of 95.8%. In the ROC category, MLP obtained the highest value of 0.950. In the next category, precision, three classifiers shared an identical value of 97.6%, whereas the others recorded 97.5%. All classifiers, except J48, had a similar result for the recall and *F*-measure categories.

5.2.1 Results in training and testing

Table 16 lists the results obtained from the 80% training and 20% testing section. In this step the classifiers were trained with 80% of the dataset, while the remaining 20% was used only for testing the model in detecting malware. The total dataset was composed of 6105 samples, 4884 samples used for training and 1221 for testing. Two classifiers NB and FT shared the best value for accuracy (95%), number of correctly (1160) and incorrectly (61) classified instances, and *F*-measure (97.2%). However, NB showed the best value (96.8%) for TPR and recall. Overall, FT was the outstanding classifier, achieving the highest values in all categories except TPR and recall in detecting unknown malware.

5.2.2 Before and after GA processes

To observe the differences before and after applying the GA process, we compare the results in predicting the class samples in WEKA. We first prepared the benign samples which were excluded from this experimental dataset to test the prediction

Table 14 Evaluation information

Value	Evaluation measure	Description	Equation
Higher value which indicates better performance	Accuracy	Correctly predicting instances as either malware or benign	$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}$
	TPR	Correctly predicting instances as malware	$\text{TPR} = \text{TP} / (\text{TP}+\text{FN})$
	Recall (similar to TPR)	Measuring the algorithm performance in identifying malicious samples	$\text{Recall} = \text{TP} / (\text{TP}+\text{FN})$
	Precision	Measuring whether the prediction is true or not	$\text{Precision} = \text{TP} / (\text{TP}+\text{FP})$
	F-measure	Measuring the weighted harmonic mean of precision and recall	$F\text{-measure} = \frac{2\times\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$
Lower value which indicates better performance	FPR	Incorrectly predicting the sample as malware when it is actually benign	$\text{FPR} = \text{FP} / (\text{FP}+\text{TN})$

Table 15 Classifier results in cross validation

Classifier	Category	Accuracy	Number of instances		FPR	TPR	ROC	Precision	Recall	*F*-measure
			Correctly classified	Incorrectly classified						
Bayes	NB	94.17%	5749	356	24.50%	**96%**	0.936	97.50%	**96%**	**96.80%**
Function	MLP	94.19%	5750	355	24.20%	**96%**	**0.950**	**97.60%**	**96%**	**96.80%**
Trees	FT	**94.22%**	**5752**	**353**	**23.80%**	**96%**	0.947	**97.60%**	**96%**	**96.80%**
	RF	94.20%	5751	354	24%	**96%**	0.946	**97.60%**	**96%**	**96.80%**
	J48	93.89%	5732	373	25.10%	95.80%	0.905	97.50%	95.80%	96.60%

effectiveness. Table 17 lists 10 benign samples downloaded from the Google Play store and scanned in VirusTotal (https://www.VirusTotal.com/) to confirm their benign status.

In the beginning, we predicted these applications using 106 features. Then machine learning predicted these samples using the GA features. Table 18 details the results before and after the GA processes.

Table 18 compares the accuracy and the time taken in machine learning before and after the GA courses. Fig. 13 graphically depicts these differences. Before the features were selected, the machine learning classifiers incorrectly predicted the benign applications as malware five times. One of the classifiers, J48, made two incorrect predictions, whereas MLP, FT, and RF made only one. Fig. 13 shows NB as

Table 16 Classifier results in cross validation

Classifier	Category	Accuracy	Number of instances		FPR	TPR	ROC	Precision	Recall	*F*-measure
			Correctly classified	Incorrectly classified						
Bayes	NB	**95%**	**1160**	**61**	22.10%	**96.80%**	0.940	97.70%	**96.80%**	**97.20%**
Function	MLP	94.90%	1159	62	22.10%	96.70%	0.953	97.70%	96.70%	**97.20%**
Trees	FT	**95%**	**1160**	**61**	**21.20%**	96.70%	**0.956**	**97.80%**	96.70%	**97.20%**
	RF	94.90%	1159	62	22.10%	96.70%	0.954	97.70%	96.70%	**97.20%**
	J48	94.70%	1157	64	23.90%	96.70%	0.916	97.50%	96.70%	97.10%

Table 17 Benign sample information

Md5	Package name	Size (byte)
cd217fd9a73b5660175acf5282e7f83a	com.schoenmueller.wiesbadenPlus	27 491 923
300ae64c25457ed12d9d4afadd90424b	com.incisivemedia.erAndroid	3 610 532
0e7c2d0422290eef1af6a80ad67e26b3	com.tester.wpswpatester	907 795
2c0e53592afdf430f1052ed7873e7124	com.appsbuilder4593	536 882
a8f5008f60c169e67feb93cfeb2f3368	com.komfo.komfo	4 085 095
66B129B07A3C4C8F82DA428FA6809528	cc.leet.free	2 191 099
f49c0c987e0187dea84033a9fcaf55a2	com.mgz.bmpkiosk	27 181 791
55f51015242995f88cc059aa368a58af	com.aor.droidedit	1 465 402
B45B49C5369FA351A54130124675EE03	com.bigint.writermd	3 942 002
53939541CF6439979F154966AB0109E6	com.aflower.weightlosssmoothies	1 403 709

Table 18 The results of the experiment before and after GA

Package name	NB		MLP		FT		J48		RF	
	Before GA	After GA	Before GA	After GA	Before GA	After GA	Before GA	After GA	Before GA	After GA
com.schoenmueller.wiesbadenPlus	B	B	B	B	B	B	B	M	B	B
com.incisivemedia.erAndroid	B	B	B	B	B	B	B	B	B	B
com.tester.wpswpatester	B	B	B	B	B	B	B	B	B	B
com.appsbuilder4593	B	B	B	B	M	B	B	M	B	B
com.komfo.komfo	B	B	B	B	B	B	B	B	B	B
cc.leet.free	B	B	B	B	B	B	B	B	B	B
com.mgz.bmpkiosk	B	B	B	B	B	B	B	M	B	B
com.aor.droidedit	M	B	M	B	M	B	M	B	B	B
com.bigint.writermd	B	B	B	B	B	B	B	B	B	B
com.aflower.weightlosssmoothies	B	B	B	B	B	B	B	B	B	B
Prediction/total benign	9/10	**10/10**	9/10	**10/10**	8/10	**10/10**	9/10	7/10	**10/10**	**10/10**
Time taken for prediction (s)	175.57	0.01	1.81	2.5	0.36	0.31	0.36	0.02	0.1	0.09

the only classifier that successfully predicted the 10 applications as benign. However, after the best were selected among 106 features, all the classifiers accurately predicted the applications, except J48, which made incorrect predictions only three times. This proves that by adopting GA to reduce the number of features, machine learning classifiers enhance the accuracy in distinguishing between benign and malware classes. Fig. 13 illustrates another significant aspect of the GA process, which is the time consumed by the classifier in conducting intelligent machine learning prediction.

Before the GA course, using 106 features, the total time taken for prediction by machine learning classifiers was 178.2 s, with MLP taking the longest time of 175.57 s (Fig. 13). Using features derived from the GA method, the classifiers successfully predicted 47 correct classes as benign with a satisfactorily short total time of 2.93 s. This shows that by applying 106 features, the prediction took a long time because the classifiers need to consider all the features in each sample, making detection much more complex. By reducing the number of features using GA, the classifiers can reduce the complexity of the features in the machine learning intrusion detection system. Having defined the number of feature differences, the next issue is the reverse engineering process in static analysis.

The machine learning steps were performed in WEKA where we reverse-engineered the samples and counted their frequency before assigning them to the simulation. However, the situation is different when the prediction takes place in the actual intrusion detection system (IDS). Practically, in the initial step of static analysis, the IDS automatically reverse-engineered each sample to obtain the code and counted each feature in machine learning classifiers. When the IDS needs to recognize 106 features in overall code, including all types of strings, including numbers, characters (e.g., double apostrophes and brackets), and symbols, this might increase the noise of the strings and lead to irrelevant data. Fig. 13 illustrates the worst accuracy in prediction and time taken in the simulation before the GA process. The predictions might achieve lower accuracy and consume much more time in real IDS. MLP took 175.57 s in the simulation process, in which the reverse engineering and counting feature processes were excluded. Therefore, in actual IDS, MLP would need more than 175.57 s to detect malware. Feature noise may increase when machine learning needs to process more than 10 samples. Therefore, by reducing the number of features with GA, we can reduce the string noise and unnecessary data, thereby contributing to the effectiveness of machine learning intelligent prediction systems.

FT achieved promising scores in cross validation as well as in training and testing phases, surpassing the other tree-based machine learning classifiers (i.e., RF and J48), despiting the fact that past analysts precluded FT from their static type examinations.

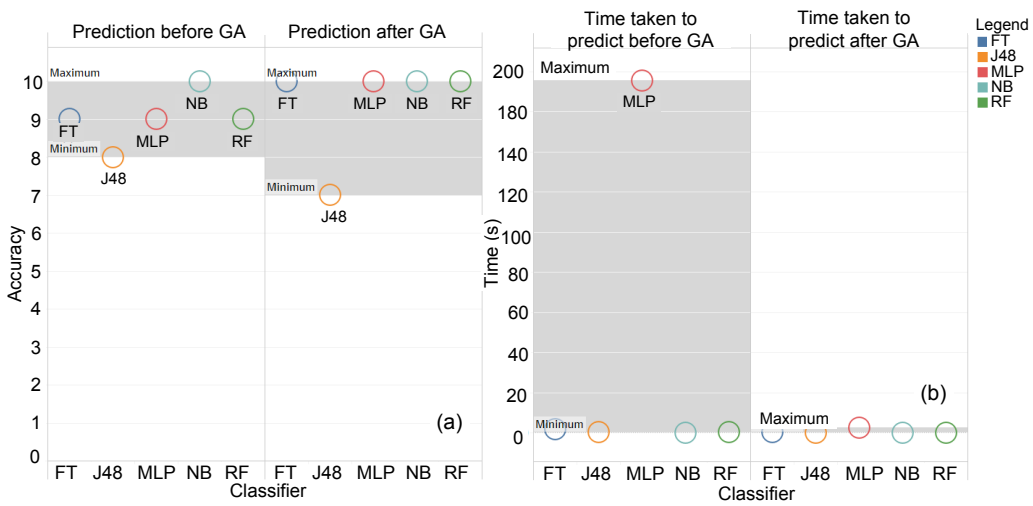

Fig. 13 A comparison of accuracy (a) and time consumption (b) in machine learning prediction

6 Comparison

Each type of analysis, either static or dynamic, applies a different method to examine malware and benign datasets. Therefore, different analyses present distinct advantages and disadvantages. To substantiate the effectiveness of the results, in this section we compare our experimental results with those of previous studies that used static and dynamic analyses with Drebin as the malware dataset. Table 19 compares the results in the accuracy category. The studies were chosen based on three criteria: (1) the dataset used was similar to Drebin; (2) the papers were published in good quality journals; (3) the research objective was to reduce the number of features to ensure relevance to malware detection. As Arp et al. (2014) did not include the TPR result, in this section we use only accuracy and FPR as the benchmarks for comparison.

In the static analysis comparison, results from the GS method were different from those of Arp et al. (2014) in selecting features. GS is based on an approach that models the natural processes of the inheritance of multiple generations and survival by fitness or adaptation to the environment. In contrast, in the joint vector space the typical patterns and combinations of the features are analyzed and identified in geometric form. However, in the comparison, our GS approach was 1% more accurate than that of Arp et al. (2014). This proved that GS performs much better than that of the joint vector space in selecting the minimum number of features. However, the frequencies of the features in their study remained unknown; therefore, in evaluating accuracy, we cannot discuss which and how many features were used for their linear support vector machine training.

On the other hand, in dynamic analysis, Smartbot (Karim et al., 2016) marked the accuracy value with 99.49%. This indicates that their method of using element tree xml in selecting features is convenient for dynamic analysis. Nevertheless, their study did not include a benign dataset as they were specifically concerned with diagnosing malware that had only command & control (C&C) features. Therefore, it was unnecessary to include benign in their dataset. In addition, although the accuracy is high in detecting malware, dynamic analysis consumes much more time compared with static analysis as the samples are classified by executing and monitoring their behavior. Moreover, our study aimed to detect all types of malware (trojan, botnet, and root exploit), whereas their study focused on detecting botnet.

Comparing the FPR for both static and dynamic analyses, our score was lower than those of Drebin and Smartbot. This demonstrates that benign samples play an important role in producing beneficial results, as more benign samples contribute to achieving a lower FPR value (a lower value indicates better performance). However, in Drebin, the FPR value corresponded to one false alarm in a limited sample of 100 applications and the exact list of features used in their experiment were unknown. Our study

Table 19 Comparison with previous studies

Type of analysis	Classifier	Accuracy	FPR	Number of datasets		Method for selecting features	Feature	Reference
				Benign	Malware			
Static	FT	95%	21.2%	550	5555	Genetic search	6 features (permission, code-based, and directory path)	This study
	Support vector machine	94%	1%	123 453	5560	Joint vector space	Set to a vector space	Drebin (Arp et al., 2014)
Dynamic	Simple logistic regression	99.49%	1%	n/a	36 for learning, 4891 for testing	Element tree xml API of python and regular expression	16 network features	Smartbot (Karim et al., 2016)

16 network features: file system activities, network connections, information leakage, started services, SMS, cryptographic operations, DNS request, HTTP traffic parameters, and unknown TCP and UDP conversations

implemented two types of evaluation, the 10-fold cross validation and training and testing, including the whole dataset. Furthermore, we listed the six features for the machine learning classifier for our experiment. In the Smartbot study, the experiment using dynamic analysis showed that the monitoring application activities consumed a lot of time, effort, and hardware resources. Our static analysis study needed fewer resources (memory and CPU) than dynamic analysis which had higher hardware and time requirements.

7 Conclusions and future work

In conclusion, Android malware was increasing along with the need to detect it. In this work, static analysis was chosen to classify malware because it required few resources, had high performance, and reviewed overall code thoroughly. As features are crucial in static analysis, it is necessary to select the smallest number of the best features to enhance accuracy (i.e., to produce an accurate predictive model) with fewer data and reduce model complexity. In our experiment GS was used to select the best of 106 features derived from permission, code-based, directory path, and system command categories. To find the best among those 106 features, we adopted GS to automatically select six exclusive features. Thereafter, we evaluated the features in five machine learning classifiers (i.e., NB, FT, J48, RF, and MLP). Among them, FT recorded the highest accuracy (94.22%) and TPR (96%), and the lowest FPR (23.8%) in 10-fold cross validation. FT achieved the highest scores in training and testing, with 95% accuracy, 1160 correctly and 61 incorrectly classified instances, 21.2% FPR, 0.956 ROC, 97.8% precision, and 97.2% F-measure. The experiment showed that FT recorded magnificent scores and outperformed other tree-based machine learning classifiers (i.e., RF and J48), even though researchers disregarded FT in their static analysis investigations.

However, the machine learning classifiers had high FPR, indicating that they incorrectly predicted some benign samples as malware because of the large disparity in the numbers of malware (5555) and benign (550) samples. By including more benign samples in the dataset, machine learning may enhance the predictive model for malware classification. Furthermore, the challenge was increased because we attempt to detect all types of malware, unlike other studies which focused on certain types of malware, such as botnet.

For future study, we recommend adding benign applications to the dataset to improve the detection results, for instance, to obtain a lower FPR and increase the TPR value. It may also be useful to explore more features in other categories in application code, such as API and network functionalities.

References

Aafer Y, Du WL, Yin H, 2013. Droidapiminer: mining API-level features for robust malware detection in Android. Proc 9th Int ICST Conf on Security and Privacy in Communication Networks, p.86-103.

Adewole KS, Anuar NB, Kamsin A, et al., 2017. Malicious accounts: dark of the social networks. *J Netw Comput Appl*, 79:41-67. https://doi.org/10.1016/j.jnca.2016.11.030

Afifi F, Anuar NB, Shamshirband S, et al., 2016. Dyhap: dynamic hybrid ANFIS-PSO approach for predicting mobile malware. *PLoS ONE*, 11(9):e0162627. https://doi.org/10.1371/journal.pone.0162627

Android, 2015. App manifest. http://developer.Android.com/guide/topics/manifest/manifest-intro.html [Accessed on Apr. 28, 2015].

Android Developers, 2015. Android security overview. Android. https://source.Android.com/devices/tech/security/ [Accessed on Sept. 1, 2015].

Anuar NB, Sallehudin H, Gani A, et al., 2008. Identifying false alarm for network intrusion detection system using hybrid data mining and decision tree. *Malays J Comput Sci*, 21(2):101-115.

Anuar NB, Papadaki M, Furnell S, et al., 2013. Incident prioritisation using analytic hierarchy process (AHP): risk index model (RIM). *Secur Commun Netw*, 6(9):1087-1116. https://doi.org/10.1002/sec.673

Apvrille A, Strazzere T, 2012. Reducing the window of opportunity for Android malware gotta catch 'em all. *J Comput Virol*, 8(1-2):61-71. https://doi.org/10.1007/s11416-012-0162-3

Arp D, Spreitzenbarth M, Malte H, et al., 2014. Drebin: effective and explainable detection of Android malware in your pocket. Proc Symp on Network and Distributed System Security, p.1-15.

Arzt S, Rasthofer S, Fritz C, et al., 2014. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android Apps. Proc 35th ACM SIGPLAN Conf on Programming Language Design and Implementation, p.259-269. https://doi.org/10.1145/2666356.2594299

Aung Z, Zaw W, 2013. Permission-based Android malware detection. *Int J Sci Technol Res*, 2(3):228-234.

Bartel A, Klein J, Le Traon Y, et al., 2012. Automatically

securing permission-based software by reducing the attack surface: an application to Android. Proc 27th IEEE/ACM Int Conf on Automated Software Engineering, p.274-277. https://doi.org/10.1145/2351676.2351722

Bird S, Klein E, Loper E, 2009. Natural language processing with Python—analyzing text with the natural language toolkit. O'Reilly Media.

Burguera I, Zurutuza U, Nadjm-Tehrani S, 2011. Crowdroid: behavior-based malware detection system for Android. Proc 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, p.15-26. https://doi.org/10.1145/2046614.2046619

Caruana R, Karampatziakis N, Yessenalina A, 2008. An empirical evaluation of supervised learning in high dimensions. Proc 25th Int Conf on Machine Learning, p.96-103. https://doi.org/10.1145/1390156.1390169

Chan PPK, Song WK, 2014. Static detection of Android malware by using permissions and API calls. Proc Int Conf on Machine Learning and Cybernetics, p.82-87. https://doi.org/10.1109/ICMLC.2014.7009096

Chang TK, Hwang GH, 2007. The design and implementation of an application program interface for securing XML documents. *J Syst Softw*, 80(8):1362-1374. https://doi.org/10.1016/j.jss.2006.10.051

Chess B, McGraw G, 2004. Static analysis for security. *IEEE Secur Priv*, 2(6):76-79. https://doi.org/10.1109/MSP.2004.111

Deshotels L, Notani V, Lakhotia A, 2014. Droidlegacy: automated familial classification of Android malware. Proc ACM SIGPLAN on Program Protection and Reverse Engineering Workshop, Article 3. https://doi.org/10.1145/2556464.2556467

Desnos A, 2015. Androguard. https://github.com/androguard/androguard [Accessed on June 29, 2015].

Díaz-Uriarte R, de Andrés SA, 2006. Gene selection and classification of microarray data using random forest. *BMC Bioinform*, 7:3. https://doi.org/10.1186/1471-2105-7-3

eBay, 2016. Online shopping. www.ebay.com [Accessed on Apr. 4, 2016].

Faruki P, Ganmoor V, Laxmi V, et al., 2013. AndroSimilar: robust statistical feature signature for Android malware detection. Proc 6th Int Conf on Security of Information and Networks, p.152-159. https://doi.org/10.1145/2523514.2523539

Feizollah A, Anuar NB, Salleh R, et al., 2013a. A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malays J Comput Sci*, 26(4):251-265.

Feizollah A, Shamshirband S, Anuar NB, et al., 2013b. Anomaly detection using cooperative fuzzy logic controller. Proc 16th FIRA RoboWorld Congress, p.220-231. https://doi.org/10.1007/978-3-642-40409-2_19

Feizollah A, Anuar NB, Salleh R, et al., 2015. A review on feature selection in mobile malware detection. *Dig Invest*, 13:22-37. https://doi.org/10.1016/j.diin.2015.02.001

Feizollah A, Anuar NB, Salleh R, et al., 2017. Androdialysis: analysis of Android intent effectiveness in malware detection. *Comput Secur*, 65:121-134. https://doi.org/10.1016/j.cose.2016.11.007

Feng Y, Anand S, Dillig I, et al., 2014. Apposcopy: semantics-based detection of Android malware through static analysis. Proc 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering, p.576-587. https://doi.org/10.1145/2635868.2635869

Firdaus A, Anuar NB, 2015. Root-exploit malware detection using static analysis and machine learning. Proc 4th Int Conf on Computer Science and Computational Mathematics, p.177-183.

Frank E, Hall MA, Witten IH, 2016. The WEKA Workbench (4th Ed.). Morgan Kaufmann. http://www.cs.waikato.ac.nz/ml/WEKA/Witten_et_al_2016_appendix.pdf

Fröhlich H, Chapelle O, Schölkopf B, 2003. Feature selection for support vector machines by means of genetic algorithm. Proc 15th IEEE Int Conf on Tools with Artificial Intelligence, p.142-148. https://doi.org/10.1109/TAI.2003.1250182

Gascon H, Yamaguchi F, Arp D, et al., 2013. Structural detection of Android malware using embedded call graphs. Proc ACM Workshop on Artificial Intelligence and Security, p.45-54. https://doi.org/10.1145/2517312.2517315

Goldberg DE, Holland JH, 1988. Genetic algorithms and machine learning. *Mach Learn*, 3(2-3):95-99. https://doi.org/10.1023/A:1022602019183

Google, 2014. Google play store. https://play.google.com/store?hl=en [Accessed on Jan. 1, 2014].

Gordon MI, Kim D, Perkins J, et al., 2015. Information-flow analysis of Android applications in droidSafe. Proc Network and Distributed System Security Symp, p.8-11.

Grace M, Zhou YJ, Wang Z, et al., 2012a. Systematic detection of capability leaks in stock Android smartphones. Proc 19th Network and Distributed System Security Symp, p.1-15.

Grace M, Zhou W, Jiang XX, et al., 2012b. Unsafe exposure analysis of mobile in-app advertisements. Proc 5th ACM Conf on Security and Privacy in Wireless and Mobile Networks, p.101-112. https://doi.org/10.1145/2185448.2185464

Grace M, Zhou YJ, Zhang Q, et al., 2012c. RiskRanker: scalable and accurate zero-day Android malware detection. Proc 10th Int Conf on Mobile Systems, Applications, and Services, p.281-294. https://doi.org/10.1145/2307636.2307663

Hall M, Frank E, Holmes G, et al., 2009. The WEKA data mining software: an update. *ACM SIGKDD Explor Newsl*, 11(1):10-18. https://doi.org/10.1145/1656274.1656278

Huang CY, Tsai YT, Hsu CH, 2013. Performance evaluation on permission-based detection for Android malware. Proc Int Computer Symp, p.111-120. https://doi.org/10.1007/978-3-642-35473-1_12

Huang JJ, Zhang XY, Tan L, et al., 2014. AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. Proc 36th Int Conf on Software Engineering, p.1036-1046. https://doi.org/10.1145/2568225.2568301

Ikinci A, Holz T, Freiling F, 2008. Monkey-spider: detecting malicious websites with low-interaction honeyclients. Proc Sicherheit-Schutz und Zuverlässigkeit, p.407-421.

Junaid M, Liu DG, Kung D, 2016. Dexteroid: detecting malicious behaviors in Android apps using reverse-engineered life cycle models. *Comput Secur*, 59:92-117. https://doi.org/10.1016/j.cose.2016.01.008

Kang H, Jang JW, Mohaisen A, et al., 2015. Detecting and classifying Android malware using static analysis along with creator information. *Int J Distr Sens Netw*, 11(6), Article 7. https://doi.org/10.1155/2015/479174

Karim A, Salleh RB, Shiraz M, et al., 2014. Botnet detection techniques: review, future trends, and issues. *J Zhejiang Univ Sci-C* (*Comput & Elcetron*), 15(11):943-983. https://doi.org/10.1631/jzus.C1300242

Karim A, Salleh R, Khan MK, 2016. Smartbot: a behavioral analysis framework augmented with machine learning to identify mobile botnet applications. *PLoS ONE*, 11(3):e0150077. https://doi.org/10.1371/journal.pone.0150077

Khatavakhotan AS, Ow SH, 2015. Development of a software risk management model using unique features of a proposed audit component. *Malays J Comput Sci*, 28(2):110-131.

Komili O, 2015. Sophos detects 100% of Android malware in independent test—for the sixth time in a row. https://blogs.sophos.com/2015/08/14/sophos-detects-100-of-Android-malware-in-independent-test-for-the-sixth-time-in-a-row/ [Accessed on Jan. 1, 2016].

Kotsiantis SB, 2013. Decision trees: a recent overview. *Artif Intell Rev*, 39(4):261-283. https://doi.org/10.1007/s10462-011-9272-4

Kotsiantis SB, Zaharakis ID, Pintelas PE, 2006. Machine learning: a review of classification and combining techniques. *Artif Intell Rev*, 26(3):159-190. https://doi.org/10.1007/s10462-007-9052-3

La Delfa GC, Monteleone S, Catania V, et al., 2016. Performance analysis of visualmarkers for indoor navigation systems. *Front Inform Technol Electron Eng*, 17(8):730-740. https://doi.org/10.1631/FITEE.1500324

Lai HJ, Tang Y, Luo HX, et al., 2011. Greedy feature selection for ranking. Proc 15th Int Conf on Computer Supported Cooperative Work in Design, p.42-46. https://doi.org/10.1109/CSCWD.2011.5960053

Lee J, Lee S, Lee H, 2015. Screening smartphone applications using malware family signatures. *Comput Secur*, 52:234-249. https://doi.org/10.1016/j.cose.2015.02.003

Lee SH, Jin SH, 2013. Warning system for detecting malicious applications on Android system. *Int J Comput Commun Eng*, 2(3):324-327. https://doi.org/10.7763/IJCCE.2013.V2.197

Liang SY, Keep AW, Might M, et al., 2013. Sound and precise malware analysis for Android via pushdown reachability and entry-point saturation. Proc 3th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, p.21-32. https://doi.org/10.1145/2516760.2516769

Lippmann R, 1987. An introduction to computing with neural nets. *IEEE ASSP Mag*, 4(2):4-22. https://doi.org/10.1109/MASSP.1987.1165576

Lu L, Li ZC, Wu ZY, et al., 2012. CHEX: statically vetting Android apps for component hijacking vulnerabilities. Proc ACM Conf on Computer and Communications Security, p.229-240. https://doi.org/10.1145/2382196.2382223

Middlemiss MJ, Dick G, 2003. Weighted feature extraction using a genetic algorithm for intrusion detection. Proc Congress on Evolutionary Computation, p.1669-1675. https://doi.org/10.1109/CEC.2003.1299873

Narudin FA, Feizollah A, Anuar NB, et al., 2016. Evaluation of machine learning classifiers for mobile malware detection. *Soft Comput*, 20(1):343-357. https://doi.org/10.1007/s00500-014-1511-6

Peiravian N, Zhu XQ, 2013. Machine learning for Android malware detection using permission and API calls. Proc 25th Int Conf on Tools with Artificial Intelligence, p.300-305. https://doi.org/10.1109/ICTAI.2013.53

Peng H, Gates C, Sarma B, et al., 2012. Using probabilistic generative models for ranking risks of Android apps. Proc ACM Conf on Computer and Communications Security, p.241-252. https://doi.org/10.1145/2382196.2382224

Punch WFIII, Goodman ED, Pei M, et al., 1993. Further research on feature selection and classification using genetic algorithms. Proc 5th Int Conf on Genetic Algorithms, p.557-564.

Rasthofer S, Arzt S, Bodden E, 2014. A machine-learning approach for classifying and categorizing Android sources and sinks. Proc Network and Distributed System Security Symp, p.1-15.

Razak MFA, Anuar NB, Salleh R, et al., 2016. The rise of "malware": bibliometric analysis of malware study. *J Netw Comput Appl*, 75:58-76. https://doi.org/10.1016/j.jnca.2016.08.022

Russon MA, 2016. Android malware discovered on Google Play has infected millions of users with spyware. http://www.ibtimes.co.uk/Android-malware-discovered-google-play-store-1553341 [Accessed on June 13, 2016].

Sahs J, Khan L, 2012. A machine learning approach to Android malware detection. Proc European Intelligence and Security Informatics Conf, p.141-147. https://doi.org/10.1109/EISIC.2012.34

Samra AAA, Yim K, Ghanem OA, 2013. Analysis of clustering technique in Android malware detection. Proc 7th Int Conf on Innovative Mobile and Internet Services in Ubiquitous Computing, p.729-733. https://doi.org/10.1109/IMIS.2013.111

Sanz B, Santos I, Laorden C, et al., 2013a. PUMA: permission

usage to detect malware in Android. Int Joint Conf CISIS'12-ICEUTE'12-SOCO'12 Special Sessions. Springer Berlin Heidelberg, p.289-298.

Sanz B, Santos I, Laorden C, et al., 2013b. Mama: manifest analysis for malware detection in Android. *Cybern Syst*, 44(6-7):469-488. https://doi.org/10.1080/01969722.2013.803889

Sarip AG, Hafez MB, Daud MN, 2016. Application of fuzzy regression model for real estate price prediction. *Malays J Comput Sci*, 29(1):15-27. https://doi.org/10.22452/mjcs.vol29no1.2

Sarma BP, Li NH, Gates C, et al., 2012. Android permissions: a perspective combining risks and benefits. Proc 17th ACM Symp on Access Control Models and Technologies, p.13-22. https://doi.org/10.1145/2295136.2295141

Schmidt AD, Bye R, Schmidt HG, et al., 2009a. Static analysis of executables for collaborative malware detection on Android. Proc IEEE Int Conf on Communications, p.1-5. https://doi.org/10.1109/ICC.2009.5199486

Schmidt AD, Schmidt HG, Batyuk L, et al., 2009b. Smartphone malware evolution revisited: Android next target? Proc 4th Int Conf on Malicious and Unwanted Software, p.1-7. https://doi.org/10.1109/MALWARE.2009.5403026

Schneider J, 2016. Cross validation. http://www.cs.cmu.edu/~schneide/tut5/node42.html [Accessed on Aug. 1, 2016].

Seo SH, Gupta A, Mohamed Sallam A, et al., 2014. Detecting mobile malware threats to homeland security through static analysis. *J Netw Comput Appl*, 38:43-53. https://doi.org/10.1016/j.jnca.2013.05.008

Shabtai A, Fledel Y, Elovici Y, 2010. Automated static code analysis for classifying Android applications using machine learning. Proc Int Conf on Computational Intelligence and Security, p.329-333. https://doi.org/10.1109/CIS.2010.77

Shabtai A, Kanonov U, Elovici Y, et al., 2012. "Andromaly": a behavioral malware detection framework for Android devices. *J Intell Inform Syst*, 38(1):161-190. https://doi.org/10.1007/s10844-010-0148-x

Sharif M, Yegneswaran V, Saidi H, et al., 2008. Eureka: a framework for enabling static malware analysis. Proc 13th Symp on Research in Computer Security, p.481-500. https://doi.org/10.1007/978-3-540-88313-5_31

Sheen S, Anitha R, Natarajan V, 2015. Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, 151:905-912. https://doi.org/10.1016/j.neucom.2014.10.004

Skylot, 2015. Jadx. https://github.com/skylot/jadx

Stein G, Chen B, Wu AS, et al., 2005. Decision tree classifier for network intrusion detection with GA-based feature selection. Proc 43rd Annual Southeast Regional Conf, p.136-141. https://doi.org/10.1145/1167253.1167288

Suarez-Tangil G, Tapiador JE, Peris-Lopez P, et al., 2014. Dendroid: a text mining approach to analyzing and classifying code structures in Android malware families. *Expert Syst Appl*, 41(4):1104-1117. https://doi.org/10.1016/j.eswa.2013.07.106

Talha KA, Alper DI, Aydin C, 2015. Apk auditor: permission-based Android malware detection system. *Dig Invest*, 13:1-14. https://doi.org/10.1016/j.diin.2015.01.001

Thomas P, 2015. Google's Android operating system dominates the smartphone market. http://finance.yahoo.com/news/google-Android-operating-system-dominates-170640913.html [Accessed on June 11, 2016].

Tropp JA, 2004. Greed is good: algorithmic results for sparse approximation. *IEEE Trans Inform Theory*, 50(10): 2231-2242. https://doi.org/10.1109/TIT.2004.834793

Walenstein A, Deshotels L, Lakhotia A, 2012. Program structure-based feature selection for Android malware analysis. Proc 4th Int Conf on Security and Privacy in Mobile Information and Communication Systems, p.51-52. https://doi.org/10.1007/978-3-642-33392-7_5

Williams G, 2010. ARFF data. http://datamining.togaware.com/survivor/ARFF_Data0.html [Accessed on Sept. 10, 2015].

Wu DJ, Mao CH, Wei TE, et al., 2012. Droidmat: Android malware detection through manifest and API calls tracing. Proc 7th Asia Joint Conf on Information Security, p.62-69. https://doi.org/10.1109/AsiaJCIS.2012.18

Yang ZM, Yang M, 2012. LeakMiner: detect information leakage on Android with static taint analysis. Proc 3rd World Congress on Software Engineering, p.101-104. https://doi.org/10.1109/WCSE.2012.26

Yerima SY, Sezer S, McWilliams G, et al., 2013. A new Android malware detection approach using Bayesian classification. Proc IEEE 27th Int Conf on Advanced Information Networking and Applications, p.121-128. https://doi.org/10.1109/AINA.2013.88

Yerima SY, Sezer S, McWilliams G, 2014a. Analysis of Bayesian classification-based approaches for Android malware detection. *IET Inform Secur*, 8(1):25-36. https://doi.org/10.1049/iet-ifs.2013.0095

Yerima SY, Sezer S, Muttik I, 2014b. Android malware detection using parallel machine learning classifiers. Proc 8th Int Conf on Next Generation Mobile Apps, Services and Technologies, p.37-42. https://doi.org/10.1109/NGMAST.2014.23

Yerima SY, Sezer S, Muttik I, 2015. High accuracy Android malware detection using ensemble learning. *IET Inform Secur*, 9(6):313-320. https://doi.org/10.1049/iet-ifs.2014.0099

Yu L, Pan ZL, Liu JJ, et al., 2013. Android malware detection technology based on improved Bayesian classification. Proc 23rd Int Conf on Instrumentation, Measurement, Computer, Communication and Control, p.1338-1341. https://doi.org/10.1109/IMCCC.2013.297

Zhang LS, Niu Y, Wu X, et al., 2013. A3: automatic analysis of Android malware. Proc 1st Int Workshop on Cloud Computing and Information Security, p.89-93. https://doi.org/10.2991/ccis-13.2013.22

Zhang T, 2009. On the consistency of feature selection using greedy least squares regression. *J Mach Learn Res*, 10: 555-568.

Zhou W, Zhou YJ, Jiang XX, et al., 2012. Detecting repackaged smartphone applications in third-party Android marketplaces. Proc 2nd ACM Conf on Data and Application Security and Privacy, p.317-326. https://doi.org/10.1145/2133601.2133640

Zhou W, Zhou YJ, Grace M, et al., 2013. Fast, scalable detection of "Piggybacked" mobile applications. Proc 2nd ACM Conf on Data and Application Security and Privacy, p.185-196. https://doi.org/10.1145/2435349.2435377

Zia T, Akhter MP, Abbas Q, 2015. Comparative study of feature selection approaches for Urdu text categorization. *Malays J Comput Sci*, 28(2):93-109.