Neural Computing and Applications
https://doi.org/10.1007/s00521-021-05875-1

S.I. : MACHINE LEARNING APPLICATIONS FOR SECURITY

A novel permission-based Android malware detection system using feature selection based on linear regression

Durmuş Özkan Şahin[1] · Oğuz Emre Kural[1] · Sedat Akleylek[1] · Erdal Kılıç[1]

Received: 23 December 2020 / Accepted: 22 February 2021
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2021

Abstract
With the developments in mobile and wireless technology, mobile devices have become an important part of our lives. While Android is the leading operating system in market share, it is the platform most targeted by attackers. Although many solutions have been proposed in the literature for the detection of Android malware, there is still a need for attribute selection methods to be used in Android malware detection systems. In this study, a machine learning-based malware detection system is proposed to distinguish Android malware from benign applications. At the feature selection stage of the proposed malware detection system, it is aimed to remove unnecessary features by using a linear regression-based feature selection approach. In this way, the dimension of the feature vector is reduced, the training time is decreased, and the classification model can be used in real-time malware detection systems. When the results of the study are examined, the highest 0.961 is obtained according to the F-measure metric by using at least 27 features.

Keywords Android malware · Malware detection · Feature selection · Static analysis · Machine learning · Linear regression

1 Introduction

Nowadays, many operations performed on personal computers can be performed on smart mobile devices. For this reason, a remarkable increase has been observed in the use of smart mobile devices in recent years. While it is reported that there are 3.5 billion smart mobile device users by 2020, this figure is expected to increase to 3.8 billion by 2021 [35]. Most of these smart devices use Android as the operating system. According to statistical report given in [36], it is determined that the Android operating system was used in 88% of the smartphones that were sold worldwide between 2009 and 2018.

Android is an open-source and Linux-based mobile operating system developed by Google. Although there are many advantages of being an open-source operating system, there are some security problems since it has a security mechanism based on permission labeling [14]. There are numerous developers who develop official and third-party apps for Android. Considering that the applications developed are uploaded on official or other application repositories without detailed examination, it stands out that there is another serious security problem. Android is the main target of malware developers due to security problems caused by both the large developer base and the operating system. According to Kaspersky, a computer and network security company, it is reported that 1,245,894 mobile malware were detected in the second quarter of 2020 [21]. This number increased by 93,232 compared to the previous quarter. It is emphasized that the vast majority of detected mobile malware threats Android devices.

Android malware are generally evaluated under these four groups, including hardware-based attacks, kernel-based attacks, hardware abstraction layer-based attacks and

✉ Durmuş Özkan Şahin
durmus.sahin@bil.omu.edu.tr

Oğuz Emre Kural
oguz.kural@bil.omu.edu.tr

Sedat Akleylek
sedat.akleylek@bil.omu.edu.tr

Erdal Kılıç
erdal.kilic@bil.omu.edu.tr

[1] Faculty of Engineering, Department of Computer Engineering, Ondokuz Mayıs University, Atakum, Samsun, Turkey

Published online: 19 March 2021

applications-based attacks [10]. These malware causes many material and non-material damages such as unauthorized action on behalf of users, making users a part of botnet attacks, collecting important data from users, producing financial gain from gathering data and damaging hardware of the device. To prevent these damages, both researchers and security companies aim to design malware detection systems with high-performance rates.

Mobile malware detection is done in three different ways: static, dynamic and hybrid [15]. Static analysis is a test technique performed over application files without the need to run the application. Dynamic analysis technique is an analysis method based on monitoring the behavior of the application by running it on a real or virtual device. The use of static and dynamic properties together is called the hybrid method. Static and dynamic analysis methods have advantages and disadvantages compared to each other. Hybrid analysis methods are being developed considering these advantages and disadvantages. Some of the studies done by adopting these techniques are as follows:

Machine learning is the modeling of systems that make predictions by making inferences from data with mathematical and statistical operations. Machine learning, which has been a popular field of study in recent years, has been used in many areas in different disciplines. Examples of these areas are the processing of medical data [23], analysis of economic data [40], software development process [34], classification of fake news [19], speech analysis or recognition [28] and the Internet of Things [31]. In addition to these studies, researchers benefit from machine learning techniques for the security of the Android operating system [25]. Static properties extracted from application files and properties extracted from application behaviors are given as input to machine learning algorithms. Thus, high performance is achieved by making Android malware detection or family classification of malware with machine learning approaches.

In [12], Burguera et al. proposed the framework named Crowdroid that enables the detection of Android applications that perform abnormal behavior. The study was based on behavior-based dynamic analysis technique. In infrastructure of the Crowdroid, a tool called "Strace" available in Linux was used to collect system calls. A large number of system calls can be evaluated with this tool. Each system call has a unique number in the Linux operating system [24]. In this way, it will be revealed which system calls an application uses. Thus, applications are classified as benign or malicious depending on whether the system calls that cause malicious activities are used or not.

In [41], Wu et al. aimed to detect Android malware with the framework they call DroidMat. Static analysis was preferred due to the cost of dynamic analysis. Various static properties were used, especially permissions and API calls. Different malware groups were created using K-means and expectation-maximization (EM) clustering algorithms. The initial number of clusters was decided through singular value decomposition. After this stage, the application was classified as malicious or benign using the K-nearest neighbors (KNN) algorithm. The proposed method was tested on 1500 benign applications and 238 malicious applications. The malicious applications were obtained from various malware families. When the results of the study were examined, the highest classification performance was achieved when K-means and KNN algorithms were used together. According to the accuracy metric, the highest performance of the DroidMat frame was 0.9787.

In [33], Sanz et al. proposed a machine learning-based Android malware detection method that use information obtained from the application permissions in the framework called as PUMA. To evaluate the PUMA framework, 1811 benign applications in various categories from Android Market and 249 unique malicious applications from VirusTotal database were used. Various permissions comparisons were made, claiming that some permissions were frequently seen in both benign and malign applications. Considering these comparisons, a feature vector was created by using the permissions requested by the application and the characteristics of the device. Feature vectors are given as input to many machine learning algorithms, and various classification achievements are obtained. When all the results are examined, it is determined that the most successful classification algorithm is Random Forest. The highest performance obtained in the study is 0.8641 according to the accuracy metric.

In [43], Yerima et al. proposed an Android malware detection method using Bayesian classification. In their studies, they performed feature ranking and selection process with mutual information calculation in order to achieve optimum classification performance. They obtained a total of 58 code-based features from applications with static code analysis. After the feature selection process, in their experiments with the Bayesian classifier, they reported that 15–20 features are sufficient to achieve optimum performance. They reported the highest accuracy they achieved in their experiments as 0.92.

In the system called DENDROID [37], Suarez-Tangil et al. analyzed the source codes of the applications with text mining and information retrieval techniques and categorize Android malware types. It was emphasized that this study was the first to perform source code analysis with text mining in mobile malware detection. While each application examined is represented as a document, the codes of the application are represented like a word in text. After each application and code are transformed into a vector, the analysis phase begins. At this stage,

dendrograms, a hierarchical clustering technique, are used to understand malware families as phylogenetic trees. With the use of dendrogram, it is aimed to remove the common features between malware families. These common features can be the similarities of certain code blocks or lines.

In [8], Arp et al. tried to detect Android malware by combining static analysis and machine learning approaches in the framework they call Drebin. By using the application's source code and the AndroidManifest.xml file, Drebin took advantage of various features such as permissions, API calls and network addresses of the application. These features were transformed into a vector space model, and Android malware detection was performed through the SVM algorithm. The Drebin method was tested on 5560 malicious and 123453 benign applications. According to the results of the study, it is seen that Drebin gives 0.939 performance. When the same dataset is classified using 10 antivirus programs, it is observed that the Drebin method gives low results from only one of these antivirus programs.

In [18], Fereidooni et al. aimed to detect Android malware with the framework they call Anastasia. Static analysis technique was preferred in the study. A tool called uniPDroid was developed using the Python programming language. Static attributes were extracted with this tool. Features that can create malicious activity such as intent filters, permissions, system commands, API calls, IMEI reading and socket opening were used. Feature selection was made by means of randomized decision tree. Classification of Android applications was provided using XGboost, AdaBoost, Random Forest, Support Vector Machine (SVM), KNN, Logistic Regression, Naive Bayes and Deep learning classification algorithms. The proposed method was tested on 18677 malware and 11187 benign applications. In the study, the best classification result is obtained by using the XGboost algorithm. This result is reported as 0.973 according to the true-positive rate.

In [2], Abdullah et al. suggested an approach that classify malware and benign applications. Selected malware applications were botnet attack family for Android operating system. The permissions requested by the applications were used as attributes in the study. After the feature vector was created, information gain, which is one of the filter-based feature selection methods, was used to select distinctive permissions. Among all the permissions, 20 permissions with the highest information gain value were selected. Authors consider that the *SET_ALARM* and *PACKAGE_USAGE_STATS* permissions do not have discrimination power among these 20 permissions. Therefore, the size of the feature vector is reduced to 18 permissions by removing these permissions in the feature vector. By implementing these steps, the number of permissions is reduced by 87%. Naive Bayes, Random Forest and C4.5 algorithms are used in the classification stage. The proposed method is tested on 1505 malicious and 850 benign applications. In the study, the best classification result is obtained by using the Random Forest algorithm. This result is reported as 0.946 according to the true-positive rate.

In [44], Yıldız and Doğru preferred application permissions from static properties as attributes. After the feature vector was created, the distinctive permissions were selected with the genetic algorithm, which is one of the wrapper feature selection methods. Naive Bayes, decision trees and SVM algorithms were used in the classification stage. The proposed method was tested on 1119 malicious and 621 benign applications. A total of 152 permissions were obtained from these applications. The best performance achieved when classifying with 152 permissions (without performing the feature selection step) was found by using the SVM algorithm, while this result was 0.959 according to the F-measure metric. The highest performance obtained in the study was found when 16 permissions were selected with the genetic algorithm and classified with the SVM algorithm. This result was 0.981 according to the F-measure metric.

Salah et al. proposed an Android malware detection method to the user by combining different static features such as URLs, hardware information, permissions and API calls [32]. By adapting the Term Frequency-Inverse Document Frequency (TF-IDF) algorithm, which is frequently used in text mining, to the malware detection domain, they proposed the frequency-based feature selection method called Feature Frequency Application Frequency (FF-FA). They obtained the result feature vector by combining the high-frequency features that passed through the feature selection with the URL_score they obtained from application URLs. In their experiments with Support Vector Machine, Logistic Regression, AdaBoost, Stochastic Gradient Descent (SGD), Latent Drichlet Allocation (LDA) algorithms, they reported that they achieved the highest result using Linear SVM with 0.99 according to the accuracy metric.

Alazab et al. proposed a classification-based Android malware detection model that combines app permissions and API calls [3]. By grouping API calls according to their frequency of use in benign and malicious applications, they revealed the most valuable API calls in terms of discrimination. According to the appearance of API calls in malicious applications, they formed three groups: ambiguous API calls (seen in almost the same number of benign and malicious applications), risky API calls (more common in malicious applications than benign applications) and disruptive API calls (seen only in malicious applications). Using the Information Gain algorithm together with the grouping process, they selected the valuable subgroup of properties and then reduced the dimensionality

of the features selected with the Term Frequency algorithm. They carried out their experiments with 5 different classifiers using a dataset containing 27891 applications in total. They reported the highest result they obtained in their experimental results with F-measure as 0.943.

Bhattacharya et al. used application permissions as attribute to detect malware specific to Android operating system [11]. The difference of the study from other permission-based malware detection approaches is the use of the rough set theory and Particle Swarm Optimization (PSO) algorithm in the feature selection phase. A better feature selection process is realized by making improvements on the particle swarm optimization algorithm. The PSO algorithm is a metaheuristic algorithm. The computational costs of metaheuristic algorithms are quite high. Therefore, feature selection with a metaheuristic algorithm can be considered as the disadvantage of the proposed method. However, since the computational cost of the PSO algorithm is less costly than other metaheuristic algorithms such as the genetic algorithm, the researchers focus on this algorithm. Apart from the datasets consisting of Android application files, different datasets are also used to evaluate the performance of the proposed method. The first of the Android datasets has 504 benign and 213 malware applications. This dataset consists of 82 permissions in total. The number of features decreases to 32 with the proposed method. When classifying with 32 permissions, a performance of 0.911875 is achieved according to the F-measure metric. The second dataset includes 2500 benign and 1150 malware applications. This dataset consists of a total of 88 permissions. The number of features decreases to 16 with the proposed method. When classifying with 16 permissions, a performance of 0.9785 is achieved according to the F-measure metric.

Yuan et al. performed both detection of malware and classification of malware according to their families [45]. The TF-IDF technique used in text classification was applied on the application permissions, and the permissions were weighted. Then, the classification step was started with various machine learning techniques. Six different classification algorithms were used in the study: Naive Bayes, Bayesian Network, C4.5, Random Tree, Random Forest and K-Nearest Neighbor. In total, 6070 benign applications and 9419 malware were used to evaluate the proposed approach. In order to fully show the accuracy of the system, the dataset was divided into various groups. The success achieved when the malware was classified according to their families is more than 0.99. It was also reported that its performance in malware detection was over 0.99.

Bai et al. proposed a framework capable of both malware detection and family classification [9]. They created the feature vector by combining the permissions obtained from the AndroidManifest.xml file and the opcode sequences obtained from the classes.dex file. They reduced dimensionality by applying the Fast Correlation-Based Filter algorithm to feature vectors, which uses a symmetrical uncertainty to measure the correlation between two features. They got the best result with 500 features in their experiments by choosing 100, 200, 300, 400 and 500 features. The CatBoost classifier, which was made open source by Yandex in 2017 [13], was used for classification processes. They reported that they achieved the best accuracy of 0.974 for malware detection and 0.9738 for family classification in their experiments with two different datasets.

Amin et al. proposed an Android malware detection system that uses end-to-end deep learning architectures based on static analysis technique [5]. Opcodes extracted from bytecodes obtained from classes.dex files in application files were given to various deep learning networks. The main algorithm used for classification was the Bidirectional Long Short-Term Memory (BiLSTM) deep learning technique. Different deep learning algorithms such as Convolutional Neural Network, Deep Belief Networks, Recurrent Neural Networks and Long Short-Term Memory were also used to make comparisons with the BiLSTM technique. Among the algorithms, the highest success was achieved with the BiLSTM technique, and this result was 0.999 according to the accuracy metric.

1.1 Motivation

When the survey studies are examined, it is seen that researchers working on Android malware detection mostly do not apply attribute selection [29, 38]. The feature selection phase has many advantages [7]. The most important of these advantages is to reduce the time spent in the training phase of classification algorithms since feature selection provides dimension reduction. In addition, determining the ideal properties in the minimum amount is very essential to increase the accuracy of the analysis and reduce the model complexity [17]. Considering that mobile devices have limited hardware and there is a need for real-time malware detection systems that can work directly on the mobile device, the feature selection phase is inevitable. For these reasons, there is a need for malware detection systems using the feature selection phase. Thus, a linear regression-based feature selection method is recommended within the developed Android malware detection system. There are two main reasons for using linear regression-based feature selection method. Firstly, the mathematical infrastructure of the linear regression method is not complex. Secondly, it can be easily integrated on the mobile device. In addition, linear regression method is used on sensors with limited resources such as mobile devices [22].

For this reason, linear regression can be easily applied in the real-time malware detection system running on the mobile device.

1.2 Contribution

The main contributions of the study are as follows:

- A novel Android malware detection system has been proposed using the linear regression-based feature selection method. The recommended system is static Android malware detection based on machine learning.
- Since feature selection is important in Android malware detection systems, a linear regression-based feature selection method is proposed in this study. To the best of our knowledge, linear regression-based attribute selection has not been used in Android malware detection so far.
- Instead of using all permissions, the most distinctive permissions are selected; thereby, the performance of classification algorithms is enhanced.
- When all permissions are used, the highest performance is achieved with the Multi-Layer Perceptron (MLP) algorithm. This result is 0.963 according to the F-measure metric. On the other hand, when only 27 permissions are used, the success achieved with the MLP algorithm is 0.961. The difference between these two results is quite small. This result shows that efficient feature selection has been made.
- In addition to the feature selection method, the study offers various permission groups that can be directly used by researchers who will work in the field of Android malware detection. Comprehensive analyses are performed by evaluating these permission groups under 7 different machine learning algorithms.

1.3 Organization

This study is organized as follows. In Sect. 2, the infrastructure and operation of the proposed Android malware detection system will be explained. In Sect. 3, datasets, classification algorithms and performance metrics used in testing the developed system will be discussed. In Sect. 4, the results obtained from the study will be given and interpreted. Finally, in Sect. 5, the general evaluation of the study will be made and information about future studies will be given.

2 Preliminaries

In this section, how to extract attributes from APK files will firstly be discussed. Next, the proposed linear regression-based feature selection method will be explained. Finally, the general structure and operation of the proposed Android malware detection system will be given.

2.1 Feature extraction

The Android Package Kit (APK) is the file format that the Android operating system uses to distribute and install apps. It contains all the items required for an application to be properly installed on the device. Many items such as application source codes, images, API calls, application permissions are included in this file. APK files can be thought of as-compressed files. For this reason, APK files must be opened in order to obtain the necessary information from the application files. In the study, files belonging to each application are opened using the AAPT2 tool [1]. Since application permissions are used as an attribute, the permissions used by the applications are obtained by accessing the AndroidManifest.xml file. Instead of using all permissions, it is ensured that the feature vector is created by considering the standard permissions that the Android operating system gives to application developers [26]. Processing the APK files and creating the feature vector are carried out as given in Algorithm 1.

Algorithm 1 Obtaining the feature vector

Input: All applications and standard permissions
Output: Feature_Vector ▷ Two dimensional array: While the number of applications is the number of rows, the number of permissions is the number of columns.

```
function CREATE_FEATURE_VECTOR(Application[ ], StandardPermission[ ])
    Feature_Vector[ ][ ] ← ∅                    ▷ Create an empty two dimensional array
    N1 ← length(Application)
    for i ← 1 to N1 do
        call AAPT2
        extract Application[i] with AAPT2
        get AndroidManifest.xml of Application[i]
        N2 ← length(StandardPermission)
        for j ← 1 to N2 do
            if StandardPermission[j] is member AndroidManifest.xml of Application[i] then
                Feature_Vector[i][j] ← 1
            else
                Feature_Vector[i][j] ← 0
            end if
        end for
    end for
    return Feature_Vector
end function
```

By running Algorithm 1, a feature vector is created according to the information on which standard permissions all applications in the dataset use or do not use. Part of the feature vector is given in Fig. 1. The feature vector is transformed into a mathematical structure by weighting as 1 if an application requests any relevant permission, and 0 if it does not. When the obtained feature vector is examined, it is determined that some permissions are not used by any application. By eliminating these permissions from the feature vector, a total of 102 permissions are evaluated. By considering all applications, the processed dataset, a part of which is given in Fig. 1, is formed. Figure 1 shows the feature vector of 1 benign and 1 malicious application for 102 permissions.

```
0,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,malware

1,0,1,1,1,0,1,1,0,0,0,0,0,0,0,1,1,0,0,
0,0,1,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,
0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,
1,0,benign
```

Fig. 1 Part of the obtained processed dataset

2.2 Linear regression-based feature selection

Linear regression is a statistical method used to model the relationship between two and more variables [27]. In the generated model to estimate the dependent variable, it is called simple regression if a simple independent variable is used as input, and multiple regression if more than one independent variable is used. In this study, application permissions correspond to the independent variables, while the dependent variable represents the type of applications. The model created is an example of multiple regression, since 102 independent variables are used. In order to create a regression model, the dependent variable must be numerical. For this reason, application labels in Fig. 1 are 0 for "malware" applications and 1 for "benign" applications.

The relationship between dependent and independent variables is calculated with regression analysis. The linear relationship between n independent variables and the dependent variable is as in Eq. 1.

$$Y = b_0 + b_1X_1 + b_2X_2 + \ldots + b_nX_n + e \tag{1}$$

In Eq. 1, Y shows the dependent variable, while $X_1, X_2, ..., X_n$ show the independent variables, in other words application permissions. The values $b_1, b_2, ..., b_n$ are the coefficients in the model, and b_0 shows the point where the line intersects the y-axis. e is defined as the error term. These coefficients are found by the least squares method. Using the least squares approach, the prediction error given in Eq. 2 is tried to be reduced to zero.

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \breve{y_i})^2 \tag{2}$$

In Eq. 2, n represents the total number of data, y_i the

observed real data, and $\breve{y}_i$ represents the estimated value of the model. Sum of Squares for Error (SSE) is the squares sum of the prediction errors. In linear regression, the SSE value is tried to be equal to zero or minimized after each coefficient is differentiated. In this way, the multiple linear regression model shown in Eq. 1 is obtained. Equation 3 shows how to obtain a multiple linear regression model consisting of 3 coefficients and 2 independent variables, i.e., $(y = a + bx_1 + cx_2)$ by using the least squares method. In Eq. 3, a, b and c represent coefficients, while x_1, x_2 and y represent independent variables and dependent variable, respectively.

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_{1,i} & \sum_{i=1}^{n} x_{2,i} \\ \sum_{i=1}^{n} x_{1,i} & \sum_{i=1}^{n} x_{1,i}^2 & \sum_{i=1}^{n} x_{1,i} \cdot x_{2,i} \\ \sum_{i=1}^{n} x_{2,i} & \sum_{i=1}^{n} x_{1,i} \cdot x_{2,i} & \sum_{i=1}^{n} x_{2,i}^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_{1,i} \cdot y_i \\ \sum_{i=1}^{n} x_{2,i} \cdot y_i \end{bmatrix} \tag{3}$$

The impact of the independent variables to the prediction of the dependent variable is determined by calculating the coefficients corresponding to each independent variable. When the multiple linear regression model is created on the obtained data, it is observed that the coefficients take values between -1 and 1. Considering Fig. 1, it is seen that the processed dataset is mostly a sparse matrix consisting of 0 values. This situation brings along the coefficients of some permissions to be 0 or close to 0. Thus, feature selection process is carried out by eliminating the permissions with coefficients close to 0 and 0 by means of Algorithm 2.

analysis. In the static analysis technique, the analysis process is performed by examining various components such as application files and source codes. For dynamic analysis, the analysis process is carried out by considering the various behaviors of the application in real or virtual environments. The advantage of the static analysis technique is that the application files are not run in real or virtual environments; thus, fast results are obtained. In addition, due to the security approach based on permission tagging in the Android operating system, it is possible to detect malware by using direct permission procedures. On the other hand, they may be vulnerable to first-day attacks and code obfuscation/changing tricks. The biggest advantage of dynamic analysis is that it is more successful than static analysis against zero-day attacks and code obfuscation/changing tricks. The main disadvantages of dynamic analysis are the creation of the necessary working environment and the longer time to detect malware than static analysis. In addition, applications in dynamic analysis technique are often run on virtual operating systems. The fact that some malware and their families perceive the virtual environment so that they do not knowingly show the malicious effect when they are run on virtual operating systems can be considered as another challenge of dynamic analysis. Considering these situations, static analysis technique is preferred in this study. Permissions, which have an important place in the security of the Android operating system, are used as attributes and tests are carried

Algorithm 2 Determining the features to be deleted

Input: Linear regression model
Output: Deleted_Features_List

```
function MAKE_FEATURE_SELECTION(RegressionModel)
    Convert the linear regression model to pairs of independent variables and coefficients.
    Store the obtained pairs in a hash data structure named Model{}   ▷ Using the hash data structure, the linear regression model is transformed into pairs in the form of independent variables and their coefficients. In the created hash data structure, while the keys are independent variables, the values corresponding to the keys are coefficients.
    N1 ← length(Model)
    for i ← 1 to N1 do
        if Model[Model.keys[i]] < 0.1 then
            if Model[Model.keys[i]] > −0.1 then
                Deleted_Features_List ← Model.keys[i]
            end if
        end if
    end for
    return Deleted_Features_List
end function
```

2.3 The proposed Android malware detection system

Three different approaches are preferred in malware detection: static analysis, dynamic analysis and hybrid

out with various machine learning approaches. Malware detection system designed for Android operating systems is given in Fig. 2. The operation of the given system is as follows:

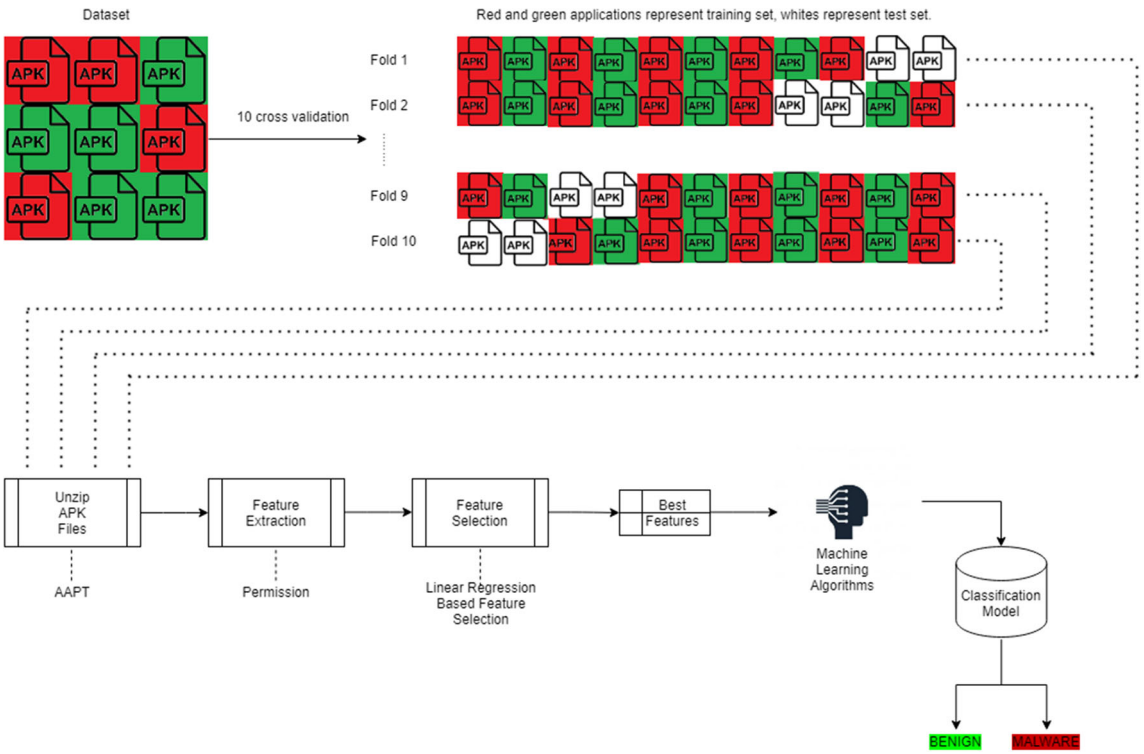

Fig. 2 The proposed Android malware detection system

Step 1: In order to specify exactly the performance of the proposed malware detection system, firstly the dataset is divided into 10 parts. In this way, 10 cross-validation is provided.
Step 2: By running Algorithm 1 on APK files, feature vector depending on application permissions is obtained.
Step 3: Multiple linear regression model is applied to the part of the feature vector that is reserved for training.
Step 4: By giving the obtained model in Step 3 as an input to Algorithm 2, low distinctive attributes are found.
Step 5: The feature selection step is performed by removing low distinctive attribute from the feature vector.
Step 6: The training part of the dataset is given to various machine learning algorithms, and a classification model is created.
Step 7: The test part of the dataset is given to the created model in Step 6, and the types of applications whose labels are unknown are estimated.
Step8: After the classification performance of the relevant cross-validation part is calculated, another cross-validation part is passed to perform all the operations described from Step 2 to 8.

By performing these 8 steps, the presented malware detection in the study is performed. The most important advantage of the proposed model is that the number of permissions required for machine learning algorithms is reduced with the linear regression-based attribute technique. The mathematics of the linear regression technique is not difficult. Therefore, it can be easily used in malware detection systems. In particular, the structure of meta-heuristic algorithms such as genetic algorithm and PSO is complex and computational costs are quite high. According to these algorithms, it is more convenient to add the proposed approach to real-time malware systems to be run on a mobile device.

Three steps are generally expected to take place in mobile malware detection systems. These are detection accuracy, real-time detection support and economical resource consumption. The detection accuracy of the proposed system reaches 96%. Since there is a size reduction technique in the proposed system, the classification model will take up less memory. In addition, the low number of attributes will significantly reduce the runtime of machine learning algorithms. Considering these situations, it is seen that the proposed malware detection system meets the needs.

3 Experimental settings

In this section, first of all, utilized dataset in the study will be mentioned. Secondly, applied machine learning algorithms to predict application types will be referred. Finally,

Table 1 Example of confusion matrix

		Predicted Class	
		Class = Benign	Class = Malware
Real class	Class = Benign	True Positive (TP)	False Negative (FN)
	Class = Malware	False Positive (FP)	True Negative (TN)

used metrics to evaluate classification performances will be given.

3.1 Used dataset

The dataset consisting of 2000 applications is used in the study. In total, 1000 of these applications are malicious, while the remaining 1000 applications are benign. Benign applications are downloaded from APKPure, which many Android users use to get apps [6]. Malicious applications are randomly selected from the Android Malware Dataset [39].

3.2 Used classification algorithms

In total, 7 different classification algorithms are used in this study. These algorithms are KNN, Naive Bayes (NB), Sequential Minimal Optimization (SMO), MLP, Random Forest (RF), C4.5 and Logistic Regression (LR). All algorithms are operated via the WEKA package [20]. The value of k is 1 in KNN. In other algorithms, default values are used in the WEKA package.

3.3 Performance measure

In the performed experiments for malware detection, 10 cross-validation test techniques are used. In this technique, the dataset is divided into 10 equal sets. A total of 10 experiments are carried out, nine of which are training and one is testing. Then the average of the obtained performance values is reported. Performance evaluations of classification algorithms used in machine learning are obtained using the confusion matrix shown in Table 1. The confusion matrix shows the relationship between real value and predicted value.

Four possible outcomes emerge from the classification of the dataset. These are true positive (TP), false negative (FN), true negative (TN) and false positive (FP). When a sample that is actually positive is correctly classified as positive, this is called TP. When the actually positive sample is misclassified as negative, it is called FN. When the actually negative sample is correctly classified as negative, it is called TN. A sample that is actually negative is considered an FP when it is incorrectly classified as positive. All these possible situations are included in the confusion matrix given in Table 1.

Considering the situations in the confusion matrix, precision and recall values emerge. Precision is the ratio of the number of true positives found by the classifier to the total of samples classified as positive. Recall is the ratio of correct positive samples to all positive samples found by the classifier. In Eqs. 4 and 5, precision and recall value are computed.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5}$$

Precision and recall values are not entirely sufficient to measure the performance of the classification algorithms. Therefore, the classification performance is calculated with the F-measure metric in Eq. 6, which is the harmonic mean of precision and recall values.

$$f - \text{measure} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{6}$$

4 Results and discussion

In this section, the results obtained from the study will be given. First, the most distinctive permissions created by applying the proposed multiple linear regression-based feature selection model on each fold of cross-validation will be listed. Secondly, the classification performance obtained in the case of not making the feature selection, and the classification performance obtained in the case of feature selection will be given comparatively. Finally, a comparison of the proposed Android malware detection system with some existing studies will be made.

In Table 2, F and TNS show fold and the total number of selection, respectively. Table 2 contains the permissions with the best discrimination power for each fold. Since the dataset is divided into 10 parts, the Android applications used in the training phase of each fold are also different. For this reason, there are differences among the permissions selected by the feature selection method on each fold before the training phase. For example, while 50 permissions are selected on Fold 1, 57 permissions are selected on Fold 2. Although the same number of permissions are selected in some folds, these permissions are not exactly the same. Fold 1–Fold 10 and Fold 2–Fold 3 are examples

Table 2 The permissions with the best discrimination power for each fold

Folds of cross-validation Permissions	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	TNS
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.ACCESS_CHECKIN_PROPERTIES	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	9
android.permission.ACCESS_COARSE_LOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.ACCESS_NETWORK_STATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.ACCESS_NOTIFICATION_POLICY	✓	✗	✗	✓	✓	✗	✓	✓	✓	✓	7
android.permission.ACCESS_WIFI_STATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.ANSWER_PHONE_CALLS	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	9
android.permission.BIND_ACCESSIBILITY_SERVICE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.BLUETOOTH	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.CALL_PHONE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.CAMERA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.CHANGE_COMPONENT_ENABLED_STATE	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	8
android.permission.CHANGE_WIFI_MULTICAST_STATE	✓	✓	✓	✗	✓	✗	✓	✓	✓	✗	7
android.permission.CHANGE_WIFI_STATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.FOREGROUND_SERVICE	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	8
android.permission.GET_PACKAGE_SIZE	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	9
com.android.launcher.permission.INSTALL_SHORTCUT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.MODIFY_AUDIO_SETTINGS	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	2
android.permission.MODIFY_PHONE_STATE	✓	✓	✓	✓	✓	✗	✓	✗	✓	✓	8
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	✓	✓	✓	✗	✗	✗	✓	✓	✓	✗	6
android.permission.PACKAGE_USAGE_STATS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.PERSISTENT_ACTIVITY	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	1
android.permission.PROCESS_OUTGOING_CALLS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_CALENDAR	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	8
android.permission.READ_CALL_LOG	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_CONTACTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_EXTERNAL_STORAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_LOGS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_PHONE_STATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.READ_SMS	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	8
android.permission.REBOOT	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	9
android.permission.RECEIVE_BOOT_COMPLETED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.RECEIVE_SMS	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	9
android.permission.RECORD_AUDIO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.REQUEST_INSTALL_PACKAGES	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	9
android.permission.SEND_SMS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
com.android.alarm.permission.SET_ALARM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.SET_DEBUG_APP	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	8
android.permission.SET_PREFERRED_APPLICATIONS	✓	✗	✓	✓	✓	✓	✓	✗	✗	✗	6
android.permission.SET_WALLPAPER	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	9
android.permission.SYSTEM_ALERT_WINDOW	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.USE_BIOMETRIC	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	9
android.permission.USE_SIP	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	4
android.permission.VIBRATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.WAKE_LOCK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.WRITE_APN_SETTINGS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.WRITE_CALENDAR	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓	8
android.permission.WRITE_EXTERNAL_STORAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10

Table 2 (continued)

Folds of cross-validation Permissions	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	TNS
android.permission.WRITE_SETTINGS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	10
android.permission.WRITE_SYNC_SETTINGS	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	2
android.permission.ACCOUNT_MANAGER	✗	✓	✓	✗	✓	✗	✗	✓	✗	✗	4
android.permission.BIND_APPWIDGET	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	1
android.permission.CALL_PRIVILEGED	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	1
android.permission.CHANGE_NETWORK_STATE	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	3
android.permission.DISABLE_KEYGUARD	✗	✓	✗	✓	✗	✗	✗	✓	✓	✗	4
android.permission.GET_ACCOUNTS	✗	✓	✓	✗	✓	✓	✗	✓	✗	✗	5
android.permission.INSTALL_LOCATION_PROVIDER	✗	✓	✓	✗	✓	✗	✗	✗	✗	✗	3
android.permission.INSTALL_PACKAGES	✗	✓	✓	✗	✓	✓	✗	✓	✗	✓	6
android.permission.READ_SYNC_SETTINGS	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	8
android.permission.REQUEST_DELETE_PACKAGES	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	2
android.permission.SET_PROCESS_LIMIT	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	2
android.permission.SET_TIME_ZONE	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	2
android.permission.TRANSMIT_IR	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓	6
android.permission.USE_FINGERPRINT	✗	✓	✓	✗	✓	✓	✗	✓	✗	✓	6
android.permission.WRITE_SECURE_SETTINGS	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓	4
android.permission.REORDER_TASKS	✗	✗	✓	✗	✓	✓	✗	✓	✓	✓	6
android.permission.STATUS_BAR	✗	✗	✓	✓	✓	✗	✗	✓	✗	✗	4
android.permission.WRITE_CALL_LOG	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	2
android.permission.DUMP	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	1
android.permission.CLEAR_APP_CACHE	✗	✗	✗	✗	✓	✗	✓	✓	✗	✗	3
android.permission.BODY_SENSORS	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	2
android.permission.BROADCAST_STICKY	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	1
android.permission.READ_PHONE_NUMBERS	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	1
com.android.voicemail.permission.WRITE_VOICEMAIL	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	1
android.permission.SET_WALLPAPER_HINTS	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	1

Fig. 3 Coefficients of permissions in the linear regression method obtained from Fold 4

```
-1.0262, -0.8425, -0.484, -0.4648, -0.4239, -0.4188, -0.2918, -0.2872, -0.2589,
-0.2441, -0.1909, -0.1692, -0.1645, -0.1582, -0.1571, -0.1498, -0.1269, -0.113,
-0.1046, -0.1043, -0.104, -0.1027, -0.102, -0.0862, -0.0841, -0.0786, -0.0598,
-0.057, -0.0569, -0.0562, -0.0542, -0.0531, -0.0478, -0.0472, -0.0466, -0.0465,
-0.0464, -0.0438, -0.0402, -0.0388, -0.0382, -0.0357, -0.0344, -0.0329, -0.0321,
-0.0298, -0.0295, -0.0294, -0.0288, -0.0283, -0.0275, -0.0229, -0.0192, -0.0186,
-0.0181, -0.0165, -0.0156, -0.0138, -0.0134, -0.0133, -0.0129, -0.0115, -0.0109,
0.0, 0.0, 0.0071, 0.0178, 0.0185, 0.0187, 0.0203, 0.0205, 0.0214, 0.0355, 0.0492,
0.0534, 0.0547, 0.055, 0.0591, 0.0621, 0.0632, 0.0659, 0.0779, 0.0822, 0.1001,
0.1003, 0.1173, 0.1214, 0.1429, 0.1545, 0.1609, 0.1637, 0.1808, 0.1905, 0.2184,
0.2876, 0.3159, 0.3341, 0.3405, 0.3557, 0.3761, 0.5236, 0.7785
```

of this situation. Although 50 permissions are selected on both Fold 1 and Fold 10, there are differences between the selected permissions. A similar situation exists for Fold 2 and Fold 3. When all the situations are taken into consideration, it is seen that the minimum number of permissions is on Fold 4 with 42, while the maximum number of permissions is on Fold 2 and Fold 3 with 57.

When the linear regression method is applied to the training set in Fold 4, the coefficients of 102 permissions are given in Fig. 3. In order to interpret the coefficients better, the coefficients are listed in ascending order. The smallest of the coefficients is −1.0262. This value is the coefficient of the *android.permission.USE_BIOMETRIC*. The largest of the coefficients is 0.7785. This value is the

Table 3 Selected some dangerous permissions

Permissions	Protection level
android.permission.ACCESS_COARSE_LOCATION	Dangerous
android.permission.CALL_PHONE	Dangerous
android.permission.CAMERA	Dangerous
android.permission.PROCESS_OUTGOING_CALLS	Dangerous
android.permission.READ_CALL_LOG	Dangerous
android.permission.READ_CONTACTS	Dangerous
android.permission.READ_EXTERNAL_STORAGE	Dangerous
android.permission.READ_PHONE_STATE	Dangerous
android.permission.RECORD_AUDIO	Dangerous
android.permission.SEND_SMS	Dangerous
android.permission.WRITE_EXTERNAL_STORAGE	Dangerous

Table 4 Results obtained from the classification of the test set belonging to Fold 4 with RF (102 permissions)

		Predicted class	
		Class = Benign	Class = Malware
Real class	Class = Benign	100	0
	Class = Malware	4	96

Table 5 Results obtained from the classification of the test set belonging to Fold 4 with RF (42 permissions)

		Predicted class	
		Class = Benign	Class = Malware
Real class	Class = Benign	100	0
	Class = Malware	3	97

coefficient of the *android.permission.SET_PREFERRED_APPLICATIONS*. There are two permissions with coefficient of 0. These are *android.permission.ACCESS_CHECKIN_PROPERTIES* and *android.permission.BROADCAST_SMS*. While the coefficients of 37 permissions are greater than 0, the coefficients of 63 permissions are less than 0. According to the feature elimination method given in Algorithm 2, 60 permissions are eliminated, and 42 permissions in Fold 4 are considered as attributes.

While some permissions are selected as attributes in only onefold, some permissions are selected as attributes in all folds. There are 8 permissions selected as attributes in only onefold. On the other hand, 27 permissions selected as attributes are seen in all folds. Some of these 27 permissions are considered in the dangerous level, according to the official developer page [26]. These permissions are given in Table 3. It is noteworthy that the proposed feature selection method finds some of the dangerous permission groups in all folds. In addition to these permissions given in Table 3, it is very important to select dangerous permissions such as *android.permission.ANSWER_PHONE_CALLS* and *android.permission.RECEIVE_SMS* seen in ninefold in most cases. Furthermore, dangerous permissions can make easier for classification algorithms to distinguish between malware and benign applications.

When the 8 permissions selected in only onefold are evaluated according to the official developer page, only the *android.permission.READ_PHONE _NUMBERS* is dangerous [26]. On the other hand, other permissions are normal, privileged or planned to be revoked in the future. Not choosing a dangerous permission such as *android.permission.READ_PHONE_NUMBERS* as an attribute in most folds can be considered as a disadvantage of the proposed attribute selection method. However, it is an advantage of the proposed method that the 7 other permissions are not among the dangerous permissions.

In order to obtain detailed classification results, different classification algorithms are run on each fold and the classification performance is recorded separately. Then, the reporting process is carried out by taking the average of the results of each classification algorithm. For example, the confusion matrix consisting of the classification of the test part of Fold 4, which consists of 102 permissions, with the RF algorithm is given in Table 4. On the other hand, the confusion matrix consisting of the classification of the test part of Fold 4, which consists of 42 permissions, with the RF algorithm is given in Table 5.

According to Table 4, 100 applications that are real benign are classified as benign as a result of classification. On the other hand, 96 of the 100 applications that are actually malicious are classified as malicious, while 4 are classified as benign. According to this information, the precision value is calculated as $\frac{100}{100+4} = 0.9615$ and the recall value as $\frac{100}{100+0} = 1$. The harmonic mean of precision and recall values is the F-measure value and is calculated as $\frac{2 \cdot 0.9615 \cdot 1}{0.9615+1} \simeq 0.98$.

Table 6 The effect of feature selection for classification performance

Classification algorithms	Without feature selection (102 permissions) F-measure	With feature selection (between 42 and 57 permissions) F-measure
NB	0.9305	0.9226
LR	0.954	0.9565
SMO	0.9655	0.9625
MLP	0.963	0.9605
KNN	0.9485	0.9515
C4.5	0.956	0.958
RF	0.9625	0.9645

According to Table 5, 100 applications that are real benign are classified as benign as a result of classification. On the other hand, 97 of the 100 applications that are actually malicious are classified as malicious, while 3 are classified as benign. According to this information, the precision value is calculated as $\frac{100}{100+3} = 0.9708$ and the recall value as $\frac{100}{100+0} = 1$. The harmonic mean of precision and recall values is the F-measure value and is calculated as $\frac{2 \cdot 0.9708 \cdot 1}{0.9708+1} \cong 0.985$.

After the extracted features are evaluated, the classification achievements obtained by using these features with various machine learning algorithms will be given. In Table 6, the average performances obtained without attribute selection are compared with the average performances where the proposed feature selection process is applied.

According to Table 6, the highest performance in case of not applying any feature selection process is obtained with SMO algorithm as 0.9655. The closest value to this algorithm is the results of MLP and RF algorithms. These results are 0.963 and 0.9625, respectively. Among all classifiers, the worst result is obtained from the NB algorithm. When the feature selection is made or not, the NB algorithm gives worse results than the other algorithms. When the feature selection is made, the highest performance is obtained from the RF algorithm. This result is 0.9645.

Table 7 The use of 42 permissions obtained in Fold 4 in all folds

Classification algorithms	With feature selection (42 permissions) F-measure
NB	0.928
LR	0.958
SMO	0.963
MLP	0.959
KNN	0.954
C4.5	0.956
RF	0.962

When the effect of feature selection on the classification algorithms is examined, the performance of RF, C4.5, KNN and LR algorithms increases, while the performance of MLP, NB and SMO algorithms decreases. As the dimension is reduced with feature selection, the learning process is diminished. In addition to diminishing the learning time, an attribute selection method is required in a way that does not reduce the performance obtained from the original dataset. In addition, it is aimed to increase the accuracy of the result model by deleting features that negatively affect performance such as meaningless and unnecessary attributes. In this way, the efficiency, applicability and understandability of the created model are ensured. Considering these situations, it is seen by many classifiers that the features which negatively affect the classification performance are removed with the proposed feature selection method. RF, C4.5, KNN and LR algorithms can be given as examples of this situation. Since the performance of these algorithms increases by using the feature selection method, with the application of feature selection, there is a slight decrease in the performance of MLP, NB and SMO algorithms. Especially in classification algorithms such as MLP and SMO with high computational costs, it is remarkable that the training process is considerably reduced and the performance does not decrease much compared to the original dataset.

In addition to the results in Table 6, some experiments are carried out to expand the results of the study. Applying of 42 permissions selected in Fold 4 to all folds is the first of these experiments. The obtained average results are given in Table 7. The second experiment is the application of 43 permissions seen at 8 and above folds to all folds. The third and fourth experiments are the application of 35 permissions selected at 9 and above folds and 27 permissions selected at only 10 folds to all folds, respectively. The average classification performance achieved by sending these permission groups to machine learning algorithms is given in Table 8.

When Table 7 is examined, the results obtained with the use of 42 permissions seem quite similar to the results given in Table 6. When the results in Table 7 are compared with the results obtained without attribute selection in Table 6, the performance of LR and KNN algorithms increases. On the other hand, the performance of NB, SMO, MLP and RF algorithms decreases. There is no change in the performance of the C4.5 algorithm. These results show that the proposed feature selection method can be applied to general.

The reason for conducting the experiments in Table 8 is to see the effect of the feature vector, which is formed by the combination of the most selected permissions, on the

Table 8 The use of some permission groups in all folds

Classification algorithms	Permissions seen at 8 and above fold (43 permissions) F-measure	Permissions seen at 9 and above fold (35 permissions) F-measure	Permissions seen at all fold (27 permissions) F-measure
NB	0.928	0.923	0.923
LR	0.96	0.961	0.958
SMO	0.964	0.961	0.96
MLP	0.957	0.959	0.961
KNN	0.954	0.953	0.953
C4.5	0.956	0.956	0.956
RF	0.96	0.959	0.96

Table 9 Comparison with previous studies

Study	Analysis	Dataset size	Classification algorithm	Feature selection method	Classification performance
PUMA [33]	Static	1811 Benign 249 Malware	RF	Permission comparison	0.8641 (Accuracy)
Yerima et al. [42, 43]	Static	1000 Benign 1000 Malware	Bayesian	Mutual information	0.92–0.94 (Accuracy)
Pehlivan et al. [30]	Static	2338 Benign 1446 Malware	RF	Cfs subset	0.949 (Accuracy)
Anastasia [18]	Static	11187 Benign 18677 Malware	XGboost	Randomized decision tree	0.973 (TP rate)
Altaher [4]	Static	100 Benign 100 Malware	RF	Information gain	0.89 (TP rate)
Abdullah et al. [2]	Static	850 Benign 1505 Malware	RF	Information gain	0.946 (TP rate)
Fatima et al. [16]	Static	20000 Benign 20000 Malware	SVM	Genetic algorithm	0.95 (Accuracy)
Yıldız and Doğru [44]	Static	621 Benign 1119 Malware	SVM	Genetic algorithm	0.981 (F-measure)
Salah et al. [32]	Static	123453 Benign 5560 Malware	SVM	FF-FA based on TF.IDF	0.99 (Accuracy)
Alazab et al. [3]	Static	13719 Benign 14172 Malware	RF	Information gain and term frequency	0.943 (F-measure)
Bhattacharya et al. [11]	Static	504 Benign 213 Malware	–	Rough set theory and PSO	0.9118 (F-measure)
Bhattacharya et al. [11]	Static	2500 Benign 1150 Malware	–	Rough set theory and PSO	0.9785 (F-measure)
Bai et al. [9]	Static	5666 Benign 5560 Malware	CatBoost	Fast correlation-based filter	0.9529 (Accuracy)
The proposed method	Static	1000 Benign 1000 Malware	MLP	Linear regression	0.961 (F-measure)

classification performance. When classifying with the feature vector consisting of 27 permissions, the highest performance is obtained from the MLP algorithm. This result is 0.961. The 0.96 value which is the closest to this result is reached by using SMO and RF algorithms. When the classification is made with the feature vector created by adding 8 permissions in ninefold to 27 permissions, these 8 permissions do not affect the performance of NB, KNN and C4.5 algorithms. Unlike these algorithms, these 8 permissions increase the performance of SMO and LR algorithms, while decreasing the performance of MLP and RF algorithms. When the classification is made with the feature vector created by adding 8 permissions seen only in ninefold and 8 permissions seen in eightfold to 27 permissions, these 16 permissions do not affect the performance of RF and C4.5 algorithms. Unlike these algorithms, these 16 permissions increase the performance of the NB, KNN, SMO and LR algorithms, while only decreasing the performance of the MLP algorithm. When the results of the three different models created in Table 8 are compared with the results obtained without attribute selection in

Table 6, it is remarkable that these three models based on majority voting occur quite successful results.

In Table 9, there is a comparison of some studies that make use of feature selection in Android malware detection and proposed study. Since the highest achievement with the least number of permits is 0.961, this value is compared with existing studies. When the linear regression-based feature selection method is compared with 13 studies, it is seen that the proposed method is better than the results of 9 studies. In [11], two different datasets are tried. While one of these results is better than our results, the other is not. The reason why the proposed method gives worse results compared to the studies of [18, 32] is probably due to the fact that it uses less static properties. However, many features are used except for application permissions in [18, 32]. Genetic algorithm is used as the feature selection method in [44]. With the genetic algorithm, a large number of attribute variations are created and the best variation that enables the classification algorithms to reach the highest performance is achieved. It is an algorithm with a very high computation cost. Although the study [44] is successful compared to linear regression-based feature selection method, its usability in real-time malware detection seems difficult. However, since the computational cost of the proposed system is lower than the genetic algorithm, it can be adapted to real-time malware systems efficiently.

Considering Table 9, researchers often use unbalanced datasets. Balanced dataset is preferred in only 6 of 13 studies compared. The proposed linear regression-based feature selection method is also run on a balanced dataset. Among all studies using a balanced dataset, the best classification result is 0.961 obtained in this study. A dataset consisting of 1000 benign and 1000 malicious software is used in both [42, 43] and this study. Therefore, the results of this study can be compared exactly with studies [42, 43]. It is stated in [42, 43] studies that the most effective feature number is between 15 and 20. When using between 15 and 20 features, the achieved performance ranges from 0.92 to 0.94. In this study, 0.961 performance is obtained by using 27 features. Considering these results, it is seen that the proposed linear regression-based feature selection method gives effective and successful results in Android malware detection.

5 Conclusions and future works

In this study, a new approach for detecting Android malware is proposed. Since the hardware of mobile devices or tablets is limited, the feature selection process is very important. Therefore, feature selection step based on linear regression is included in the proposed malware detection system. In this way, insignificant permissions are eliminated from the feature vector so that the efficiency, applicability and understandability of the created model are ensured. In addition to the feature selection method, various permission groups and feature vectors are offered to researchers who will work in this field. When the results are examined, it is observed that the proposed malware detection system has achieved remarkable success by requesting a small number of application permissions. In future studies, the usability of different feature selection methods in this field will be investigated. In addition, it is aimed to increase the classification performance by using deep learning techniques as well as classical machine learning algorithms. Finally, hyperparameter selections are not made in the used machine learning algorithms. The future focus will be on hyperparameter selections to increase classification performance.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Android asset packaging tool (AAPT2) (2020). https://developer.android.com/studio/command-line/aapt2
2. Abdullah Z, Saudi MM, Anuar NB (2017) Abc: android botnet classification using feature selection and classification algorithms. Adv Sci Lett 23(5):4717–4720
3. Alazab M, Alazab M, Shalaginov A, Mesleh A, Awajan A (2020) Intelligent mobile malware detection using permission requests and API calls. Future Gener Comput Syst 107:509–521
4. Altaher A (2016) Classification of android malware applications using feature selection and classification algorithms. VAWKUM Trans Comput Sci 10(1):1–5
5. Amin M, Tanveer TA, Tehseen M, Khan M, Khan FA, Anwar S (2020) Static malware detection and attribution in android bytecode through an end-to-end deep system. Future Gener Comput Syst 102:112–126
6. Apkpure android application store. (2020)APKPure.com
7. Arauzo-Azofra A, Aznarte JL, Benítez JM (2011) Empirical study of feature selection methods based on individual feature evaluation for classification problems. Expert Syst Appl 38(7):8170–8177
8. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K, Siemens C (2014) Drebin: effective and explainable detection of android malware in your pocket. Netw Distrib Syst Secur Symp (NDSS) 14:23–26
9. Bai H, Xie N, Di X, Ye Q (2020) Famd: A fast multifeature android malware detection framework, design, and implementation. IEEE Access 8:194729–194740
10. Bhat P, Dutta K (2019) A survey on various threats and current state of security in android platform. ACM Comput Surv (CSUR) 52(1):1–35
11. Bhattacharya A, Goswami RT, Mukherjee K (2019) A feature selection technique based on rough set and improvised PSO

algorithm (PSORS-FS) for permission based detection of android malwares. Int J Mach Learn Cybern 10(7):1893–1907
12. Burguera I, Zurutuza U, Nadjm-Tehrani S (2011) Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices, pp. 15–26
13. Dorogush AV, Ershov V, Gulin A (2018) Catboost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363
14. Enck W, Ongtang M, McDaniel P (2009) Understanding android security. IEEE Secur Priv 7(1):50–57
15. Faruki P, Bharmal A, Laxmi V, Ganmoor V, Gaur MS, Conti M, Rajarajan M (2014) Android security: a survey of issues, malware penetration, and defenses. IEEE Commun Surv Tutor 17(2):998–1022
16. Fatima A, Maurya R, Dutta MK, Burget R, Masek J (2019) Android malware detection using genetic algorithm based optimized feature selection and machine learning. In: 2019 42nd International conference on telecommunications and signal processing (TSP), pp. 220–223. IEEE
17. Feizollah A, Anuar NB, Salleh R, Wahab AWA (2015) A review on feature selection in mobile malware detection. Dig Investig 13:22–37
18. Fereidooni H, Conti M, Yao D, Sperduti A (2016) Anastasia: android malware detection using static analysis of applications. In: 2016 8th IFIP International conference on new technologies, mobility and security (NTMS), pp. 1–5 . https://doi.org/10.1109/NTMS.2016.7792435
19. Hakak S, Alazab M, Khan S, Gadekallu TR, Maddikunta PKR, Khan WZ (2021) An ensemble machine learning approach through effective feature extraction to classify fake news. Future Genera Comput Syst 117:47–58
20. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. ACM SIGKDD Explor Newslett 11(1):10–18
21. IT threat evolution Q2 2020. Mobile statistics. (2020)https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/
22. Khandoker TUI, Huang D, Sreeram V (2011) A low complexity linear regression approach to time synchronization in underwater networks. In: 2011 8th International conference on information, communications and signal processing, pp. 1–5. IEEE
23. Latif J, Xiao C, Tu S, Rehman SU, Imran A, Bilal A (2020) Implementation and use of disease diagnosis systems for electronic medical records based on machine learning: A complete review. IEEE Access 8:150489–150513
24. Linux system call table. (2020) https://thevivekpandey.github.io/posts/2017-09-25-linux-system-calls.html
25. Liu K, Xu S, Xu G, Zhang M, Sun D, Liu H (2020) A review of android malware detection approaches based on machine learning. IEEE Access 8:124579–124607
26. The official site for Android app developers. (2020) https://developer.android.com/reference/android/Manifest.permission.html
27. Montgomery DC, Peck EA, Vining GG (2012) Introduction to linear regression analysis, vol 821. Wiley, New Jersey
28. Padmanabhan J, Johnson Premkumar MJ (2015) Machine learning in automatic speech recognition: a survey. IETE Tech Rev 32(4):240–251
29. Pan Y, Ge X, Fang C, Fan Y (2020) A systematic literature review of android malware detection using static analysis. IEEE Access 8:116363–116379
30. Pehlivan U, Baltaci N, Acartürk C, Baykal N (2014) The analysis of feature selection methods and classification algorithms in permission based android malware detection. In: 2014 IEEE symposium on computational intelligence in cyber security (CICS), pp. 1–8. IEEE
31. RM SP, Maddikunta PKR, Parimala M, Koppu S, Reddy T, Chowdhary CL, Alazab M (2020) An effective feature engineering for DNN using hybrid PCA-GWO for intrusion detection in iomt architecture. Computer Communications
32. Salah A, Shalabi E, Khedr W (2020) A lightweight android malware classifier using novel feature selection methods. Symmetry 12(5):858
33. Sanz B, Santos I, Laorden C, Ugarte-Pedrero X, Bringas PG, Álvarez G (2013) Puma: Permission usage to detect malware in android. In: International joint conference CISIS-12-ICEUTE 12-SOCO 12 Special Sessions, pp. 289–298. Springer
34. Srinivasan K, Fisher D (1995) Machine learning approaches to estimating software development effort. IEEE Trans Softw Eng 21(2):126–137
35. Number of smartphone users from 2016 to 2021. (2020)https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
36. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018. (2020)https://www.statista.com/statistics/266136/
37. Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Blasco J (2014) Dendroid: a text mining approach to analyzing and classifying code structures in android malware families. Exp Syst Appl 41(4):1104–1117
38. Wang W, Zhao M, Gao Z, Xu G, Xian H, Li Y, Zhang X (2019) Constructing features for detecting android malicious applications: Issues, taxonomy and directions. IEEE Access 7:67602–67631
39. Wei F, Li Y, Roy S, Ou X, Zhou W (2017) Deep ground truth analysis of current android malware. In: International conference on detection of intrusions and malware, and vulnerability assessment, pp. 252–276. Springer
40. Wei X, Chen W, Li X (2021) Exploring the financial indicators to improve the pattern recognition of economic data based on machine learning. Neural Comput Appl 33:723–737
41. Wu DJ, Mao CH, Wei TE, Lee HM, Wu KP (2012) Droidmat: android malware detection through manifest and api calls tracing. In: 2012 Seventh Asia joint conference on information security, pp. 62–69. IEEE
42. Yerima SY, Sezer S, McWilliams G (2014) Analysis of Bayesian classification-based approaches for android malware detection. IET Inf Secur 8(1):25–36
43. Yerima SY, Sezer S, McWilliams G, Muttik I (2013) A new android malware detection approach using bayesian classification. In: 2013 27th International conference on advanced information networking and applications (AINA), pp. 121–128. IEEE
44. Yildiz O, Dogru IA (2019) Permission-based android malware detection system using feature selection with genetic algorithm. Int J Softw Eng Knowl Eng 29(02):245–262
45. Yuan H, Tang Y, Sun W, Liu L (2020) A detection method for android application security based on TF-IDF and machine learning. Plos one 15(9):e0238694

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