IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 10, 2021, accepted January 17, 2022, date of publication January 27, 2022, date of current version February 8, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3146363

LinRegDroid: Detection of Android Malware Using Multiple Linear Regression Models-Based Classifiers

DURMUŞ ÖZKAN ŞAHİN, SEDAT AKLEYLEK, AND ERDAL KILIÇ
Department of Computer Engineering, Ondokuz Mayıs University, 55139 Samsun, Turkey
Corresponding author: Durmuş Özkan Şahin (durmus.sahin@bil.omu.edu.tr)

This study was supported by Ondokuz Mayıs University BAP under Grant PYO.MUH.1908.22.001.

ABSTRACT In this study, a framework for Android malware detection based on permissions is presented. This framework uses multiple linear regression methods. Application permissions, which are one of the most critical building blocks in the security of the Android operating system, are extracted through static analysis, and security analyzes of applications are carried out with machine learning techniques. Based on the multiple linear regression techniques, two classifiers are proposed for permission-based Android malware detection. These classifiers are compared on four different datasets with basic machine learning techniques such as support vector machine, k-nearest neighbor, Naive Bayes, and decision trees. In addition, using the bagging method, which is one of the ensemble learning, different classifiers are created, and the classification performance is increased. As a result, remarkable performances are obtained with classification algorithms based on linear regression models without the need for very complex classification algorithms.

INDEX TERMS Ensemble learning, linear regression, machine learning, malware analysis, permission-based android malware detection, static analysis.

I. INTRODUCTION

When the first mobile phones were considered, generally speaking or short message transactions were carried out with mobile phones in daily life. However, with mobile phones used today, remarkable transactions such as banking transactions, social media use, and personal data storage take place. Because of these essential processes, mobile devices are the main target of malware developers.

Android is an open-source Linux-based mobile operating system. Since it is open-source and free, mobile device manufacturers prefer this operating system on their devices. Therefore, the majority of the market consists of Android devices. According to Statista's data, 30% of the market in the fourth quarter of 2010 consisted of the Android operating system. In the second quarter of 2018, 88% of the market was Android operating systems [1]. In addition to Android being an open-source operating system, it is very flexible for users that applications are provided to devices such as other stores or third-party applications apart from the official application stores. For this reason, Android is frequently preferred by many people around the world.

Although applications from unofficial application repositories or third-party application developers are very advantageous for users, it should not be ignored that some of these applications are malware. Apps in official app repositories are carefully analyzed and published in app repositories. However, malware is common even in official application repositories [2]. In the research conducted by Wang *et al.*, more than 6 million applications downloaded from 17 application stores are evaluated [3]. While 16 of these stores are widely used in China, the first place is Google Play. In general, it is revealed that Google Play is more reliable than other application stores. However, it is possible to see malware in almost all stores [3].

While 1 million new malware were detected in the first six months of 2015, 1.85 million new malware were detected in the first six months of 2019 [4]. Despite all the precautions, there is a remarkable increase in the number of malicious software. For this reason, both researchers and companies working on computer security offer new approaches for detecting mobile malware. In this study, a machine learning-based

The associate editor coordinating the review of this manuscript and approving it for publication was Senthil Kumar.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Android malware detection system is developed, in which application permissions, which have an important place in Android security, are used as attributes. After an application is installed on the device, many permissions are requested from the user. While the application is running in the background, the application can show its malicious feature in line with the permissions given by the user. Therefore, users should pay attention to the requested permissions. In this study, the permissions requested by the applications are evaluated with machine learning models, and it is decided whether the application is malware or not.

A. RELATED WORKS

In recent years, many studies have been conducted to detect Android malware using machine learning or deep learning approaches. Detection methods differ according to the way in which the features used in machine learning or deep learning approaches are obtained. These are generally static, dynamic, and hybrid analysis techniques [5]. In dynamic analysis, features for machine learning approaches are obtained by running applications on a real or virtual device. In static analysis, features are extracted for machine learning approaches without running applications. Since applications are run in dynamic analysis, it is challenging to create the necessary infrastructure. However, they are successful against zero-day attacks. In static analysis, the process is quite fast since applications are not run. In addition to static and dynamic analysis techniques, there is also a hybrid analysis approach. In this approach, features obtained from static and dynamic methods are used together. Some Android malware detection systems using static, dynamic, and hybrid analysis approaches are as follows:

In [6], it was classified 2000 malicious applications consisting of 18 families according to their families. Applications were processed through the Cuckoo Sandbox, extracting the most distinctive behavioral features that distinguish malicious families from each other. The obtained features were given to a system called online machine learning, and classification of malware according to their families is carried out. In the experiments, all of the applications in 7 classes were classified correctly. The class with the lowest performance rate was determined as the android.trojan.smskey family.

In [7], a malware detection system based on dynamic analysis was proposed. In total, more than 12000 applications were evaluated. While 4289 of these applications were malicious, 8371 of them were benign. Malicious applications were obtained from the Drebin dataset, while benign applications were downloaded from Google Play. System calls were extracted dynamically and used as attributes for machine learning algorithms. The generation of system calls was handled by the sandbox. What applications do on the operating system was recorded in log files. Thus, the behaviors of each application were formed chronologically. While accessing system calls, malware was not allowed to affect these calls. In this way, the situation of changing the behavior of malicious software was also eliminated. Thanks to this feature, the proposed system was resistant to simple obfuscation techniques, which are often seen in malware. Feature vectors were created by processing the obtained log files. In the last step, these feature vectors were evaluated with machine learning approaches, and classification of benign and malicious software was carried out. In the classification phase, machine learning techniques such as support vector machines (SVM), random forest (RF), LASSO, and ridge regularization were used. The best performance was obtained from the RF algorithm.

In [8], the authors offered two different approaches based on static analysis by making use of machine learning approaches. In the first approach, application permissions were extracted with static analysis. In the second approach, source code analysis was done with the bag-of-words model. It was stated that the computational cost of the first approach is relatively low compared to the second approach. A large number of experiments were carried out using both clustering and classification algorithms. C4.5 decision tree, RF, Bayes networks, sequential minimal optimization (SMO), repeated incremental pruning (JRip), logistic regression were some of the algorithms used. In addition, models based on bagging techniques were developed by combining classification algorithms. Machine learning algorithms were run on the M0Droid dataset, which consists of 200 malicious and 200 benign Android applications. The highest performance obtained in the permission-based approach was obtained with the SMO algorithm. This performance was 0.879 based on the f-measure metric. By trying different bagging techniques, this success was increased up to 0.894. In the source code analysis, the highest performance was achieved with the SMO algorithm. This performance was 0.951 according to the f-measure metric. By trying different bagging techniques, this success was increased up to 0.9560.

In [9], the authors provided the detection of Android malware with a dynamic analysis technique. In the dynamic analysis phase, the behavior of the applications was analyzed by considering the system calls. The proposed architecture was called ANDROIDETECT. ANDROIDETECT was a machine learning-based Android malware detection method that enables instant attack detection. The classification result of the proposed detection method has a low false-positive rate, thanks to the creation of effective feature vectors. Feature vectors were created by extracting the system call function. Classification algorithms then evaluated these feature vectors. The study used two different classification algorithms, naive Bayes (NB) and J48 decision trees. Experiments were carried out with 100 benign and 100 malicious applications. The result from the NB classifier is 0.825 according to the f-measure metric. In contrast, the result obtained from the J48 classifier is 0.86 according to the f-measure metric.

In [10], 1233 Android malware were classified according to types. In total, 28 different types of Android malware were classified according to their types. Application permissions are given as input to machine learning algorithms. Some permissions were under the very dangerous group, while some

permissions were under the relatively less dangerous group. To digitize these differences and improve the performance of classification algorithms, the authors proposed a technique they call an Extremely Randomized Tree. The proposed method also satisfied the feature selection task. Six different classification algorithms were used in the study. These are SVM, ID3 decision trees, RF, neural networks, nearest neighbor, and bagging algorithms. The best classification result is obtained with the RF algorithm. The classification result obtained with the RF is 95.97%.

In [11], a permission-based Android malware detection system based on machine learning algorithms was presented. With the method called significant permission identification (SIGPID), instead of using all permissions, it was provided to choose the permissions that will facilitate the separation of malicious software from malicious software. With the proposed method, 135 permissions were reduced to 22 permissions. When classification was made with 22 permissions, more successful and faster results are obtained. In addition, it was emphasized that over 90% classification success was achieved with the SVM in the study.

In [12], 31185 benign and 15336 malicious Android applications were used. Permissions and API calls were extracted as attributes in the malware detection system called MalPat. RF algorithm was used in the classification phase of the study. When the experimental results were examined, a classification success rate of 98.24% was obtained according to the f-measure.

In [13], an Android malware detection system based on deep neural networks (DNN) was proposed. Application permissions extracted using the static analysis technique were used as attributes. In the study, extensive experiments compared deep neural networks with many traditional machine learning approaches. In the experiments, 7622 applications are evaluated. While 6661 of these applications were malicious applications, 961 of them were benign applications. 80% of the dataset was split for training and 20% for testing. The highest performance was achieved with deep neural networks. This result was reported as 0.9820 according to the f-measure metric. It was observed that deep neural networks give better results than traditional machine learning approaches.

B. MOTIVATION

In [14], the authors reported how linear regression works in permission-based Android malware detection. In the study, the error rates of the prediction values produced by the regression techniques were compared without performing the classification process. The linear regression technique comes into prominence with less error rate when compared to methods that give good results, such as multilayer perceptron, support vector machine-based regression, and additive regression. This study's main motivation is to investigate how a classifier based on linear regression will yield results in a permission-based malware detection system since it produces fewer errors than well-known techniques.

There are many studies that convert and use linear regression techniques to classifiers. In [15], iris, statlog (heart), and balance scale datasets in the UCI Machine Learning Repository are classified with the classifier obtained from the linear regression technique. Compared to the linear regression technique KNN, higher performances are obtained [15]. In [16], a hybrid classification algorithm is proposed using artificial neural networks and multiple linear regression. The proposed technique is tested on datasets with different problems such as the Fisher iris dataset, Forensic glass dataset, Japanese credit dataset, and Pima Indian Diabetes dataset. Linear regression is also frequently used in face recognition or classification problems [17]–[20]. In general, it is seen that the linear regression model is used in many pattern recognition and machine learning problems. However, when the important survey studies in the context of Android malware detection based on machine learning are examined [21]–[23], no malware detection system based on a linear regression model is found. This study uses the linear regression model to detect malware detection with two different rule-based classification algorithms. The proposed classification models have two important advantages. First, the proposed models are more successful than the KNN and NB algorithms. The second is that a simple decision-maker can be obtained by only needing the linear regression equation. In this way, a classifier that can work directly on mobile devices can be used. The resource consumption of mobile devices and battery consumption are directly related. In other words, as resource consumption increases, mobile devices consume more energy. Therefore, the resource consumption of mobile devices will not be adversely affected as the proposed classifier is quite simple. As a result, the proposed detection system will work without straining the mobile device.

C. CONTRIBUTION

The main contributions of the study can be summarized as follows:

- This study is the first comprehensive in Android malware detection that uses a linear regression model to detect Android malicious applications to the best of our knowledge.
- A general framework for Android malware detection based on permissions is proposed.
- Considering the equations produced as a result of linear regression, two different rule-based classifiers are created. The malware detection system obtained from the first rule is LinRegDroid1, and the malware detection system obtained from the second rule is LinRegDroid2.
- Obtained classification algorithms are compared with KNN, NB, SVM, decision trees (DT), and bagging of decision trees (Bagging-DT) using 10-fold cross-validation technique. The proposed classifiers are pretty successful compared to KNN and NB techniques. When the proposed approaches are compared with classification algorithms that give good results, such as SVM and decision trees, the results are comparable.

- The most successful classification algorithms are used together with the bagging technique based on majority voting to increase the performance of the classification algorithms.
- In linear regression, equations and coefficients are created according to the least-squares method. In addition to the least-squares technique, it is investigated how the obtained equation yield results when the coefficients are given random values.
- Experiments are carried out with two different evaluation metrics using classification algorithms with varying structures on four different datasets.

D. ORGANIZATION

The remaining parts of the study are organized as follows: In Section II, data preprocessing and classifiers based on linear regression techniques are discussed. In addition, bagging techniques created by combining the most successful classifiers are mentioned. In Section III, the datasets used, classification algorithms used, and the metrics used to evaluate the performance of the classifiers are given. In Section IV, the results from the study are detailed. In Section V, a general evaluation is made, and future works are discussed.

II. METHODOLOGY

This section consists of three subsections. In Section II-A, the structure of APK files and how permissions are extracted with the static analysis technique are discussed. The proposed classification approaches are detailed in Section II-B. In Section II-C, permission-based Android malware detection architecture is given.

A. DATA PREPROCESSING AND PREPARATION

Android Package Kit (APK) is known as the package file format used by the Android operating system to distribute and install mobile applications. Therefore, APK files are needed in the Android operating system. APK files can be thought of as compressed files. In general, these files include application source codes, application permissions, image and video files in applications.

Android applications are usually written using the Java programming language. Then, Java source codes are compiled and converted into byte codes. Considering computers with a Windows or Linux-based operating system on which the Java virtual machine is installed, these compiled byte codes are converted into a structure that can be run on the relevant operating system. However, byte codes cannot be run directly in the Android operating system. Therefore, bytecodes are converted to executable Dalvik bytecodes by performing one more operation on bytecodes. Thus, these Dalvik bytecodes can now be run with the help of the Dalvik Virtual Machine. As a result, the written applications are run on the device. Extracting information from APK files is the reverse of compilation. This process is called decompilation.

The process of extracting information without running APK files is called static analysis. When any APK file is extracted, some folders and files appear, as seen in Figure 1. These obtained files or folders are processed, and static properties are revealed. In this study, application permissions are accessed by evaluating AndroidManifest.xml files extracted from APK files. This is done via the Android Asset Packaging Tool (AAPT2) tool [24]. Figure 2 shows the permissions in the AndroidManifest.xml file. By combining application permissions, feature vectors are created. All the permissions obtained are checked in the AndroidManifest.xml files of the applications. If the relevant permission is included in the AndroidManifest.xml file of an application, the feature vectors of the applications are created as in Table 1 by assigning a value of 1, and if not, 0. Table 1 shows the feature vectors of a malicious application and a benign application randomly taken from the M0Droid dataset.

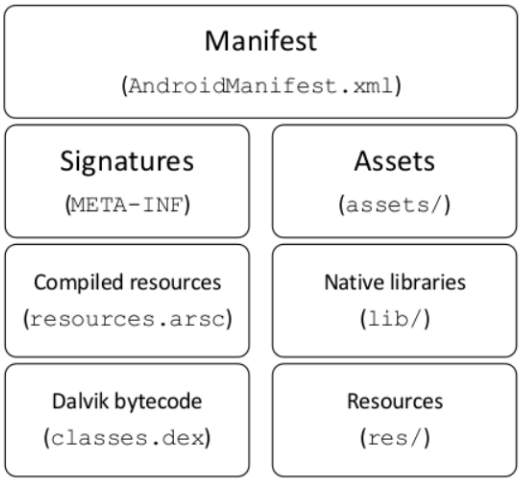


FIGURE 1. Extraction of APK files.

TABLE 1. An example of the feature vector.

$Permission_1$	$Permission_2$	$Permission_3$	...	$Permission_{75}$	$Permission_{76}$	class
1	1	1	...	0	0	benign
0	0	0	...	1	0	malicious

B. PROPOSED CLASSIFIERS

We firstly give classifiers obtained from linear regression in Section II-B1. Then, we show combining the best algorithms according to the bagging technique in Section II-B2.

1) LINEAR REGRESSION-BASED CLASSIFIERS

The linear regression technique is a frequently used method in solving estimation problems. It is based on the theory that samples in the same class belong to the same linear subspace and can be represented by a linear equation [17]. Equation 1 shows the simple linear regression model.

$$y = \beta_0 + \beta_1 X + \varepsilon \tag{1}$$

In Equation 1, y is called the dependent variable, and X is called the independent variable. The point where the line intersects the y-axis is β_0, while β_1 represents the regression

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.android.alarm.permission.SET_ALARM" />
<uses-permission android:name="android.permission.VIBRATE" />
```

FIGURE 2. Some permissions appearing in AndroidManifest.xml.

coefficient. Finally, ε represents the error of the obtained estimate. Equation 1 is known as simple linear regression since it contains only the independent variable X. If there is more than one independent variable affecting the Equation 1, it is called multiple linear regression. The multiple regression model is given in Equation 2. Considering the Equality 2, there are many independent variables consisting of $X_1, X_2, \ldots, X_n$.

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon \tag{2}$$

Considering the problem addressed in this study, while attributes, in other words, permissions, represent the independent variable, y represents the class of an application. A multiple linear regression model is needed because a large number of application permissions are used as attributes. In Table 1, the type of application is shown as benign or malicious. Since the systems of equations are solved in linear regression, operations are performed by using 1 instead of benign and 0 instead of malicious.

Suppose a dataset consists of N applications and M permissions $(p_1, p_2, \ldots, p_M)$ obtained from these applications. A system of equations can be created when there is a linear relationship between permissions and applications, as shown in Equation 3.

$$\begin{cases} y'_1 = \beta_0 + \beta_1 p_{1,1} + \beta_2 p_{1,2} + \ldots + \beta_M p_{1,M} \\ y'_2 = \beta_0 + \beta_1 p_{2,1} + \beta_2 p_{2,2} + \ldots + \beta_M p_{2,M} \\ y'_3 = \beta_0 + \beta_1 p_{3,1} + \beta_2 p_{3,2} + \ldots + \beta_M p_{3,M} \\ \vdots \\ y'_N = \beta_0 + \beta_1 p_{N,1} + \beta_2 p_{N,2} + \ldots + \beta_M p_{N,M} \end{cases} \tag{3}$$

In Equation 3, $y'_1, y'_2, \ldots, y'_N$ represents the result of linear combinations of permissions $(p_1, p_2, \ldots, p_M)$. β_i shows the effect of permissions on $y'_1, y'_2, \ldots, y'_N$ values. In Equation 3, it is aimed to find the appropriate $\beta_i (1 \leq i \leq M)$ parameter for linear regression model. The actual class values $(y_1, y_2, \ldots, y_N)$ will be approximately equal to $y'_1, y'_2, \ldots, y'_N$ values.

The mean square error is usually used to measure the quality of the linear regression model. The smaller the mean square error, the closer the linear regression model will produce to the actual value. Therefore, in order to obtain a good quality regression model, it is necessary to make the mean square error of the model as small as possible. Hence, quality regression models are created by finding the most appropriate β_i parameter. Equation 4 shows how the sum of squares of errors (SSE) is calculated.

$$\begin{aligned} SSE &= \sum_{j=1}^{N} (y_j - y'_j)^2 \\ &= \sum_{j=1}^{N} (y_j - \beta_0 - \sum_{k=1}^{M} \beta_k p_{j,k})^2 \end{aligned} \tag{4}$$

In order to minimize the SSE function obtained in Equation 4, the partial derivatives of this function with respect to each of its $\beta_i (1 \leq i \leq M)$ unknowns must be taken. Since it is aimed to minimize the error, the result of partial derivatives is equal to 0. Equation 5 shows partial derivatives.

$$\begin{aligned} \frac{\partial SSE}{\partial \beta_0} &= \frac{\partial \sum_{j=1}^{N} (y_j - \beta_0 - \sum_{k=1}^{M} \beta_k p_{j,k})^2}{\partial \beta_0} = 0 \\ \frac{\partial SSE}{\partial \beta_1} &= \frac{\partial \sum_{j=1}^{N} (y_j - \beta_0 - \sum_{k=1}^{M} \beta_k p_{j,k})^2}{\partial \beta_1} = 0 \\ &\vdots \\ \frac{\partial SSE}{\partial \beta_M} &= \frac{\partial \sum_{j=1}^{N} (y_j - \beta_0 - \sum_{k=1}^{M} \beta_k p_{j,k})^2}{\partial \beta_M} = 0 \end{aligned} \tag{5}$$

Equation 6 is obtained when partial derivatives are applied according to each of the β_i unknowns in Equation 5. A matrix and Y vector shown in Equation 6 can be obtained directly from the dataset. Since A and Y are known, the vector β can be found with $A^{-1}Y$ operation. Each element of the resulting β vector corresponds to β_i unknowns, respectively. Eq. (6), as shown at the bottom of the next page.

As a result of the calculation of the regression coefficients (β_i) in Equation 2, a linear regression model will be obtained. When the feature vectors obtained from the applications are given to this model, as shown in Table 1, the class value of the application belonging to the feature vector is determined. As a result of this calculation, the class value of the relevant application emerges, not the class label. Since the classification problem is handled in this study, Algorithm 1 and Algorithm 2 are applied separately to the obtained class value, resulting in two different results. The first of these results is called LinRegDroid1, while the second

is called LinRegDroid2. Both Algorithm 1 and Algorithm 2 provide the classification of applications by processing the result of linear regression equation according to simple rules. In Algorithm 1, if the class values obtained as a result of linear regression are greater than or equal to 0.5, a value of "1" is assigned to the class label, in other words, a benign label. Otherwise, the class label of the application is assigned as "0", that is, the malicious label. A similar rule is included in Algorithm 2. In Algorithm 2, it is determined whether the class values obtained as a result of linear regression are closer to 0 or 1. If the class value is closer to 0, the label of the relevant application is assigned a "0", that is, a malicious label. Otherwise, the application is labeled with "1", that is, benign.

2) BAGGING OF THE BEST CLASSIFIERS

Models based on ensemble learning are generally constructed in two different ways. The first of these is the bagging method, while the second is the boosting method. The advantages and disadvantages of these methods relative to each other are analyzed in detail by Dietterich [25]. In this study, classification models based on ensemble learning are created using bagging techniques. Models based on the bagging method are generally created, as shown in Figure 3. As seen in Figure 3, n random sub-datasets are created from the dataset used for training. If classifiers are trained on each of these n subsets, n different models will emerge. In the last case, when a sample in the test set is tested with these n models, n classification results are calculated. The class of the tested sample is determined by majority voting. For example, suppose there is a problem with two classes (*label1*, *label2*). Let a tested sample be classified as *label1* by k models and *label2* by l model (where $k + l = n$). If the k value is greater than l, the tested sample will be classified as *label1*. Otherwise, the sample tested will be classified as *label2*. By applying the same steps to all samples in the test data, the classes of the samples in the test data are estimated.

In this study, two different ensemble learning models are created based on the bagging technique. In the first model built, the training part of the dataset is randomly divided into five subsets. Then, the linear regression model is applied to each sub-part created. As a result, five different models emerge. Each application in the testing phase is passed through these models. Then, the types of applications are estimated by majority voting. This method is called Ensemble-1. The infrastructure of Ensemble-1 includes the decision-maker obtained from Algorithm 2. The second ensemble learning model created is called Ensemble-2. Here, the training part of the dataset is randomly divided into five subsets. Then, linear-SVM is applied to two of the formed parts while DT is applied to two of them. A linear regression model is applied to the remaining part. First, each application in the testing phase is evaluated with these five models. Then, the types of applications are estimated by majority voting. While creating both Ensemble-1 and Ensemble-2, care is taken to ensure that the number of subsets is odd. The reason

$$
\begin{aligned}
\beta_0 N + \beta_1 \sum_{i=1}^{N} p_{i,1} + \beta_2 \sum_{i=1}^{N} p_{i,2} + \ldots + \beta_M \sum_{i=1}^{N} p_{i,M} &= \sum_{i=1}^{N} y_i \\
\beta_0 \sum_{i=1}^{N} p_{i,1} + \beta_1 \sum_{i=1}^{N} p_{i,1}^2 + \beta_2 \sum_{i=1}^{N} p_{i,1}p_{i,2} + \ldots + \beta_M \sum_{i=1}^{N} p_{i,1}p_{i,M} &= \sum_{i=1}^{N} p_{i,1}y_i \\
\beta_0 \sum_{i=1}^{N} p_{i,2} + \beta_1 \sum_{i=1}^{N} p_{i,1}p_{i,2} + \beta_2 \sum_{i=1}^{N} p_{i,2}^2 + \ldots + \beta_M \sum_{i=1}^{N} p_{i,2}p_{i,M} &= \sum_{i=1}^{N} p_{i,2}y_i \\
&\vdots \\
\underbrace{\beta_0 \sum_{i=1}^{N} p_{i,M} + \beta_1 \sum_{i=1}^{N} p_{i,1}p_{i,M} + \beta_2 \sum_{i=1}^{N} p_{i,2}p_{i,M} + \ldots + \beta_M \sum_{i=1}^{N} p_{i,M}^2 = \textstyle\sum_{i=1}^{N} p_{i,M}y_i}
\end{aligned}
$$

$$
\Downarrow
$$

$$
\underbrace{\begin{bmatrix}
N & \sum_{i=1}^{N} p_{i,1} & \sum_{i=1}^{N} p_{i,2} & \cdots & \sum_{i=1}^{N} p_{i,M} \\
\sum_{i=1}^{N} p_{i,1} & \sum_{i=1}^{N} p_{i,1}^2 & \sum_{i=1}^{N} p_{i,1}p_{i,2} & \cdots & \sum_{i=1}^{N} p_{i,1}p_{i,M} \\
\sum_{i=1}^{N} p_{i,2} & \sum_{i=1}^{N} p_{i,1}p_{i,2} & \sum_{i=1}^{N} p_{i,2}^2 & \cdots & \sum_{i=1}^{N} p_{i,2}p_{i,M} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\sum_{i=1}^{N} p_{i,M} & \sum_{i=1}^{N} p_{i,1}p_{i,M} & \sum_{i=1}^{N} p_{i,2}p_{i,M} & \cdots & \sum_{i=1}^{N} p_{i,M}^2
\end{bmatrix}}_{A}
\underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_M \end{bmatrix}}_{\beta}
=
\underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} p_{i,1}y_i \\ \sum_{i=1}^{N} p_{i,2}y_i \\ \vdots \\ \sum_{i=1}^{N} p_{i,M}y_i \end{bmatrix}}_{Y}
\tag{6}
$$

Algorithm 1 Determining Class Labels With LinRegDroid1

Input: *TestData*[ ][ ] and β[ ] represent the dataset and the regression coefficients, respectively.

Output: *ClassificationLabel*[ ] represents the predicted class labels of each tested application.

```
function Classify(TestData[ ][ ], β[ ])
    N1 ← number_of _applications
    N2 ← number_of _permissions
    results[ ] ← ∅
    for i ← 1 to N1 do
        sum ← 0
        for j ← 1 to N2 do
            sum ← sum + TestData[i][j] * β[j]
        end for
        results[i] ← β[0] + sum
    end for
    ClassificationLabel[ ] ← ∅
    for i ← 1 to N1 do
        if results[i] >= 0.5 then
            ClassificationLabel[i] ← 1
        else
            ClassificationLabel[i] ← 0
        end if
    end for
    return ClassificationLabel[ ]
end function
```

Algorithm 2 Determining Class Labels With LinRegDroid2

Input: *TestData*[ ][ ] and β[ ] represent the dataset and the regression coefficients, respectively.

Output: *ClassificationLabel*[ ] represents the predicted class labels of each tested application.

```
function Classify(TestData[ ][ ], β[ ])
    N1 ← number_of _applications
    N2 ← number_of _permissions
    results[ ] ← ∅
    for i ← 1 to N1 do
        sum ← 0
        for j ← 1 to N2 do
            sum ← sum + TestData[i][j] * β[j]
        end for
        results[i] ← β[0] + sum
    end for
    ClassificationLabel[ ] ← ∅
    for i ← 1 to N1 do
        if abs(0 − results[i]) < abs(1 − results[i]) then
            ClassificationLabel[i] ← 0
        else
            ClassificationLabel[i] ← 1
        end if
    end for
    return ClassificationLabel[ ]
end function
```

for this is that the equality situation does not occur in the majority voting.

C. PERMISSION-BASED ANDROID MALWARE DETECTION SYSTEM

The permission-based malware detection system that provides the classification of malware is given in Figure 4. Figure 4 is applied step by step to ensure that malicious software is separated from benign software. First, datasets are created. Details of the datasets used are discussed in Section III-A. In this study, a 10-fold cross-validation technique is used. First, the dataset is divided into ten parts. Nine of these parts are used for training, and 1 for testing. In each iteration, the parts reserved for testing are changed, and all applications on the dataset are tested. This process is repeated ten times to calculate the average performance. After the datasets are created, the permissions are obtained from the applications by applying a preprocessing step on the applications. After this stage, each application is converted into a feature vector. Obtaining the feature vector is very important for machine learning algorithms. If the feature vectors specific to these algorithms are not given as input, these algorithms cannot calculate. Classification models are created by providing feature vectors to classification algorithms. Preprocessing steps are also applied to the applications reserved for testing, and they are converted into feature vectors. By introducing these vectors to classification models, the types of applications are predicted.

III. EXPERIMENTAL SETTINGS

This section consists of three subsections. In Section III-A, the datasets used are mentioned. In Section III-B, we give more details about compared classifiers with which the proposed classification approaches. In Section III-C, we present the metrics used to measure the performance of classification algorithms.

A. DATASETS USED

In this study, four different datasets are used. The first dataset is shared by Ali Dehghantanha, one of the authors of study M0Droid [26]. In this dataset, there are 200 benign and 200 malicious applications. When the data preprocessing step in Section II-A is applied to this dataset, 76 native permissions are extracted as attributes. The second dataset is AMD. There are 1000 malicious and 1000 benign applications in this dataset. The malicious applications in this dataset are obtained from [27], [28]. Benign applications are downloaded from the APKPure app store [29]. We extract 102 native permissions from the AMD dataset. The third dataset is shared in [30], [31]. There are 558 applications in total in this dataset. Half of these applications are benign, while the remaining half are malicious. There are 330 attributes in this dataset, consisting of native and custom permissions. Finally, the

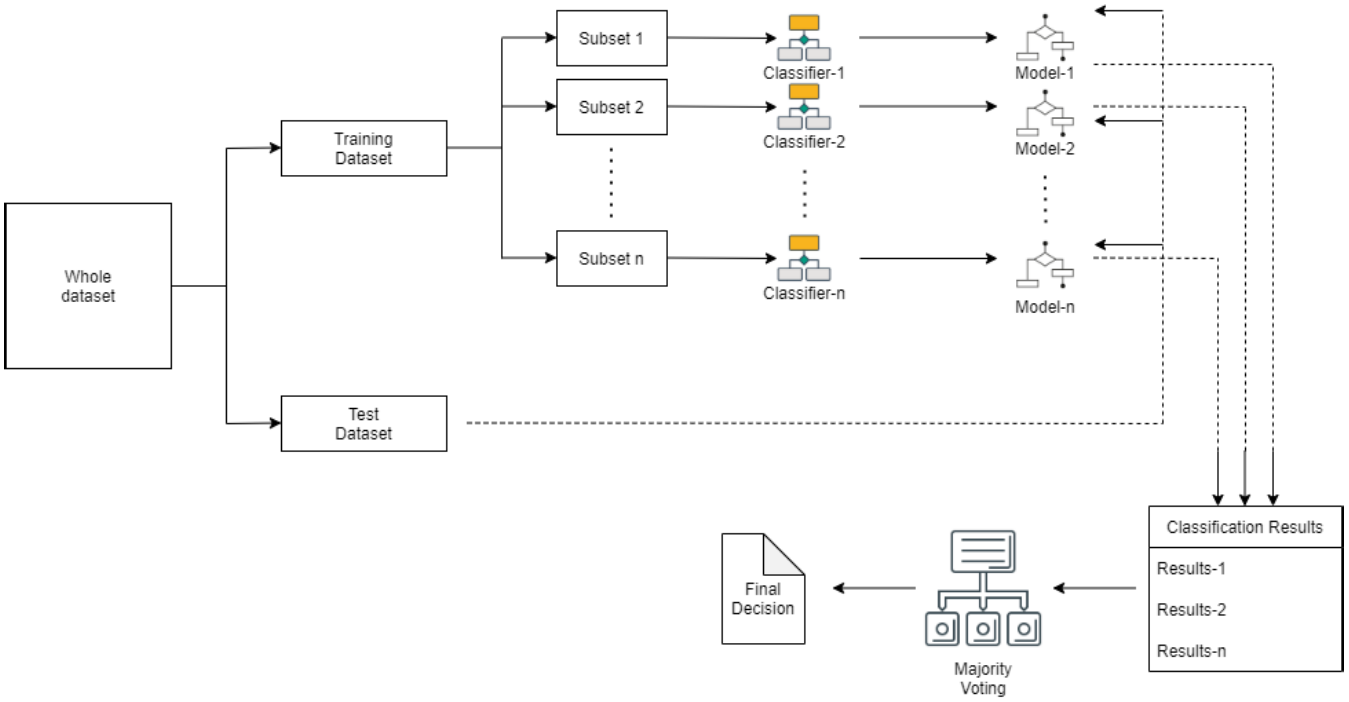


FIGURE 3. The general framework of bagging models.

fourth dataset is shared in [13]. There are 7622 applications in total in this dataset. While 6661 of these applications are malicious, 961 of them are benign. This dataset contains 349 attributes consisting of native and custom permissions.

B. CLASSIFIERS USED IN COMPARISON

Basically, five different machine learning techniques are used to compare the classification algorithms based on linear regression proposed in this study. These are KNN, NB, SVM, DT, and Bagging-DT algorithms. In addition, some of these algorithms are preferred among the algorithm combination methods based on the proposed bagging technique. MATLAB R2016 is used for these algorithms. By trying different parameters in NB and SVM algorithms, the results obtained from these algorithms are expanded. The algorithms used in the study and their parameters are detailed in Table 2.

According to Table 2, default parameters are used in the DT algorithm. In the KNN algorithm, classification is performed by choosing the k value as 1. In the NB algorithm, classification is made using two different distributions. The first of these is multinomial distribution (mn), while the second is multivariate multinomial distribution (mvmn). Two different kernel functions are used in the SVM algorithm. These are linear and radial basis functions. Finally, the Bagging-DT algorithm is implemented with a total of five trees.

C. PERFORMANCE MEASURE

The confusion matrix is frequently used to measure the performance of machine learning approaches. An example of a confusion matrix is shown in Table 3. Some of the information indicated in Table 3 are as follows:

TP: It is the number of samples that are actually in the "+" class but classified with "+" as a result of the classification.

TN: It is the number of samples that are actually in the "−" class but classified with "−" as a result of the classification.

FP: It is the number of samples that are actually in the "−" class but classified with "+" as a result of the classification.

FN: It is the number of samples that are actually in the "+" class but classified with "−" as a result of the classification.

TABLE 2. Algorithms and their parameters.

Classification Algorithm	Function Name of Algorithm in MATLAB	Parameters of Algorithm
KNN	fitcknn	'NumNeighbors', 1
mn-NB	naivebayes.fit	'Distribution', 'mn'
mvmn-NB	naivebayes.fit	'Distribution', 'mvmn'
linear-SVM	fitcsvm	'KernelFunction', 'Linear'
rbf-SVM	fitcsvm	'KernelFunction', 'rbf'
DT	fitctree	default parameter
Bagging-DT	TreeBagger	$number\_of\_trees = 5$, 'OOBPredictorImportance', 'On'

TABLE 3. Example of confusion matrix.

		Predicted Class	
		-	+
Real Class	-	True Negative (TN)	False Positive (FP)
	+	False Negative (FN)	True Positive (TP)

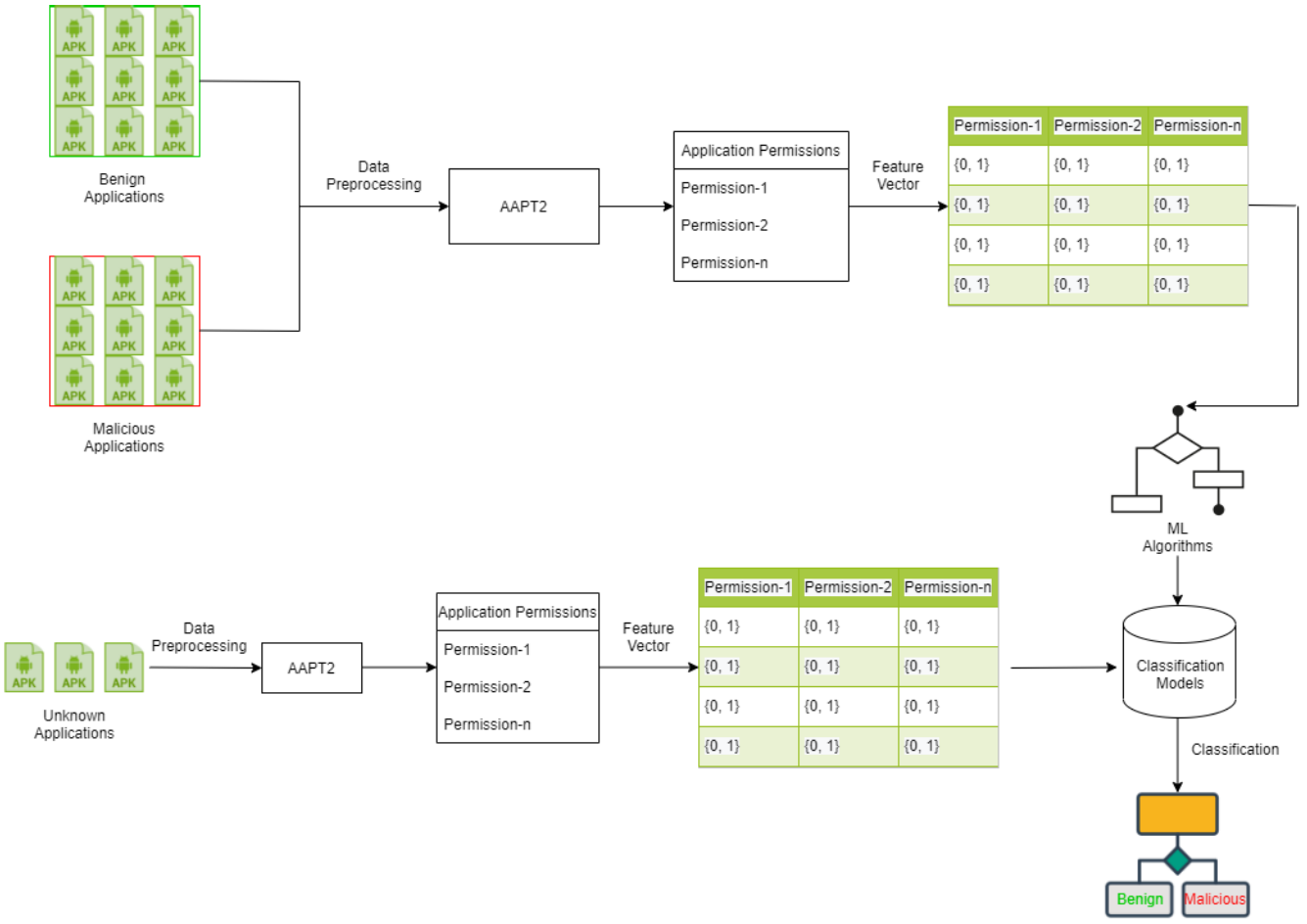


FIGURE 4. The general framework of permission-based Android malware detection system.

By using TP, TN, FP, and FN values, accuracy in Equation 7, precision in Equation 8, and recall metrics in Equation 9 are given.

$$accuracy = \frac{TN + TP}{TN + FN + FP + TP} \tag{7}$$

$$precision = \frac{TP}{FP + TP} \tag{8}$$

$$recall = \frac{TP}{FN + TP} \tag{9}$$

Comparison with the accuracy metric may not be sufficient in experiments performed on unbalanced datasets. For this reason, it is more accurate to compare with the f-measure metric, which is the harmonic mean of precision and recall values. Equation 10 contains the mathematical representation of the f-measure metric. Considering the Table 3, two different values of precision, recall, and f-measure metrics, consisting of (+) and (−) classes, emerge. For this reason, classification algorithms are evaluated by averaging the values obtained for both classes.

$$f - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{10}$$

IV. RESULTS AND DISCUSSIONS

This section consists of two subsections. In Section IV-A, the results obtained from the study are detailed and interpreted. In Section IV-B, the results of some studies in the literature are compared with the results obtained from this study.

A. EXPERIMENTAL RESULTS

In this section, we interpret the results obtained from the datasets. Table 4 contains the results from the AMD dataset. These results are the average of 10-fold cross-validation. On the AMD dataset, LinRegDroid1 and LinRegDroid2 show 0.9560 performance according to both the accuracy and the f-measure metric. While the result obtained with the KNN algorithm is 93.6% according to the accuracy metric, it is 0.9359 according to the f-measure metric. LinRegDroid1 and LinRegDroid2 provide 2% improvement over the KNN algorithm. The mn-NB and mvmn-NB classifiers demonstrate 0.9001 and 0.9320 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model show 2% to 5% higher performance than the NB algorithm. linear-SVM and rbf-SVM methods give 0.9655 and 0.9278 performances, respectively,

TABLE 4. Results from the AMD dataset.

	accuracy(%)	f_measure
LinRegDroid1	95.6	0.956
LinRegDroid2	95.6	0.956
KNN	93.6	0.9359
mvmn-NB	93.2	0.932
mn-NB	90.05	0.9001
rbf-SVM	92.8	0.9278
linear-SVM	96.55	0.9655
DT	95.6	0.956
Bagging-DT	96.5	0.965
Ensemble-1	96.4	0.964
Ensemble-2	96.95	0.9695

TABLE 5. Results from the Lopez' dataset.

	accuracy(%)	f_measure
LinRegDroid1	91.87	0.9187
LinRegDroid2	91.87	0.9187
KNN	83.75	0.8359
mvmn-NB	88.12	0.8811
mn-NB	85.62	0.8553
rbf-SVM	91.25	0.9123
linear-SVM	93.75	0.9375
DT	92.5	0.925
Bagging-DT	91.5	0.915
Ensemble-1	91.25	0.9123
Ensemble-2	92.5	0.925

according to the f-measure metric. The approaches based on the proposed linear regression model are 3% more successful than the rbf-SVM model. However, these models show 1% less performance compared to the linear-SVM model. LinRegDroid1, LinRegDroid2, and DT models show the same results on the AMD dataset. In order to make a fair comparison on the existing Bagging-DT and Ensemble-1 and Ensemble-2 models, the training set is randomly divided into five parts, and bagging techniques are compared. Bagging-DT, Ensemble-1, and Ensemble-2 show nearly identical performances on the AMD dataset. Considering all the results, the highest performance achieved is from the Ensemble-2 model. This result is 0.9695 according to both the accuracy metric and the f-measure metric.

Table 5 presents the results obtained from Lopez's dataset. This dataset has quite a lot of permissions given the number of apps. Despite 558 applications, there are 330 permissions. This makes it difficult to construct an excellent linear regression model in general. Therefore, it is a complex dataset to classify. LinRegDroid1 and LinRegDroid2 give 0.9187 performance in Lopez's dataset according to the accuracy metric and the f-measure metric. While the result obtained with the KNN algorithm is 83.75% according to the accuracy metric, it is 0.8359 according to the f-measure metric. LinRegDroid1 and LinRegDroid2 provide 8% improvement over the KNN algorithm. The mn-NB and mvmn-NB classifiers yield 0.8553 and 0.8811 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model show 3% to 6% higher performance than the NB algorithm. linear-SVM and rbf-SVM methods give 0.9375 and 0.9123 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model show similar results with the rbf-SVM model. However, these models show 2% less performance when compared to the linear-SVM model. The approaches based on the proposed linear regression model show 1% less performance when compared to the DT model. Bagging-DT, Ensemble-1, and Ensemble-2 bagging techniques give lower results than the main classifiers on this dataset. For example, the result obtained with the DT model is 0.925 according to the f-measure metric, while the result obtained with the Bagging-DT is 0.9150 according to the f-measure metric. A similar situation is seen in the results of Ensemble-1 and Ensemble-2. Considering all the results, the highest performance obtained is from the linear-SVM model. This result is 0.9375 according to both the accuracy metric and the f-measure metric.

Table 6 shows the results obtained from the M0Droid dataset. On the M0Droid dataset, LinRegDroid1 and LinRegDroid2 give 82.942% performance according to the accuracy metric, and 0.8287 according to the f-measure metric. While the result obtained with the KNN algorithm is 82.69% according to the accuracy metric, it is 0.8258 according to the f-measure metric. Both LinRegDroid1, LinRegDroid2 and KNN produce similar results. The mn-NB and mvmn-NB classifiers have 0.7733 and 0.7765 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model show 5% higher performance than the NB algorithm. linear-SVM and rbf-SVM methods give 0.8619 and 0.8673 performances, respectively, according to the f-measure metric. Unlike the datasets of AMD and Lopez, the rbf kernel function produces more successful results in this dataset. The approaches based on the proposed linear regression model give lower results than both the rbf-SVM model and the linear-SVM model. In addition, the approaches based on the proposed linear regression model show 4% less performance when compared to the DT model. Bagging-DT, Ensemble-1 and Ensemble-2 bagging techniques give higher results than the main classifiers on this dataset. For example, the result obtained with the DT model is 0.8619 according to the f-measure metric, while the result obtained with the Bagging-DT is 0.8712 according to the f-measure metric. A similar situation is seen in the results of Ensemble-1 and Ensemble-2. The result obtained with the LinRegDroid2 model is 0.8287 according to the f-measure metric, while the result obtained with the Ensemble-1 is 0.8348 according to the f-measure metric. Considering all the results, the highest performance is obtained from the Ensemble-2 model. This result is 89.22% according to the accuracy metric and 0.8915 according to the f-measure metric.

Table 7 shows the results obtained from Arslan's dataset. Unlike other datasets, the accuracy and f-measure metrics on this dataset are quite different because this dataset is unbalanced. On this dataset, LinRegDroid1 and LinRegDroid2

TABLE 6. Results from the M0Droid dataset.

	accuracy(%)	f_measure
LinRegDroid1	82.942	0.8287
LinRegDroid2	82.942	0.8287
KNN	82.69	0.8258
mvmn-NB	77.923	0.7765
mn-NB	77.429	0.7733
rbf-SVM	86.962	0.8673
linear-SVM	86.212	0.8619
DT	86.212	0.8619
Bagging-DT	87.205	0.8712
Ensemble-1	83.74	0.8348
Ensemble-2	89.22	0.8915

TABLE 7. Results from the Arslan' dataset.

	accuracy(%)	f_measure
LinRegDroid1	96.69	0.9172
LinRegDroid2	96.69	0.9172
KNN	96.54	0.9126
mvmn-NB	93.96	0.8571
mn-NB	94.74	0.8667
rbf-SVM	94.86	0.8617
linear-SVM	97.76	0.9470
DT	97.63	0.9443
Bagging-DT	97.01	0.9249
Ensemble-1	96.91	0.9229
Ensemble-2	98.53	0.9662

give 96.69% performance according to the accuracy metric and 0.9172 according to the f-measure metric. While the result obtained with the KNN algorithm is 96.54% according to the accuracy metric, it is 0.9126 according to the f-measure metric. The mn-NB and mvmn-NB classifiers yield 0.8667 and 0.8571 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model show 6% higher performance than the NB algorithm. linear-SVM and rbf-SVM methods give 0.9470 and 0.8617 performances, respectively, according to the f-measure metric. The approaches based on the proposed linear regression model are 5% more successful than the rbf-SVM model. However, these models show 3% less performance when compared to the linear-SVM model. Also, the approaches based on the proposed linear regression model show 3% less performance when compared to the DT model. On this dataset Ensemble-1, and Ensemble-2 except Bagging-DT bagging techniques, gives higher results than the main classifiers. However, Bagging-DT gives a lower performance. For example, the result obtained with the DT model is 0.9443 according to the f-measure metric, while the result obtained with the Bagging-DT is 0.9249 according to the f-measure metric. On the other hand, the result obtained with the LinRegDroid2 model is 0.9172 according to the f-measure metric, while the result obtained with the Ensemble-1 is 0.9229 according to the f-measure metric. Considering all the results, the highest performance is obtained from the Ensemble-2 model. While this result is 98.53% according to the accuracy metric, it is 0.9662 according to the f-measure metric.

It is seen that the classifiers based on the linear regression model created according to the results obtained from the datasets generally give good results. It is also shown that in permission-based malware detection, data in the same class will belong to the same linear subspace and can be expressed by a linear equation. Since there is a linear relationship between the dataset and the samples, it is possible to make predictions for other samples through the linear regression technique. Finally, it should not be ignored that the obtained bagging techniques also give good results. In the creation of bagging techniques, since the datasets are relatively small, the training parts of the datasets are randomly divided into five parts. It is possible to obtain higher performances by creating more subsets in larger datasets. Also, in this study, different regression models are created by assigning random values to the regression coefficients. Findings of randomly generated models are included in Remark 1.

Remark 1: The regression coefficients obtained in this study generally vary between −1 and 1. 10000 regression models are created by giving random values between −1 and 1 to the regression coefficients. However, the error rates of random models are higher than the actual model. For example, in experiments on the AMD dataset, the Pearson correlation coefficient of the actual regression model is 0.8836. The result of the best randomly generated model is 0.8694 according to the Pearson correlation coefficient. Only 3429 of these random models have Pearson correlation coefficient above 0.80. Better models can be created by developing smart search strategies instead of brute-force searching.

B. COMPARISON WITH PREVIOUS WORKS

In this subsection, the results obtained will be compared with some results in the literature. Table 8 compares the results of existing studies with the results obtained in this study. While making comparisons, not only static analysis is taken into account, but also the results obtained from some dynamic and hybrid studies are included. Comparisons are made with the highest performances reported in existing studies and the classification algorithms in which these performances are obtained. In this study, since a permission-based Android malware detection system is proposed, permission-based models will be evaluated among themselves first. A general comparison will then be made.

According to Table 8, there are 5 studies that only use permissions as an attribute. The highest performance obtained from these studies is obtained from the AndroAnalyzer [13] as 0.9820 according to the f-measure metric. Using the same dataset, the result of 0.9662 is obtained according to the f-measure metric with the Ensemble-2 technique. Our result is approximately 2% lower than [13]. However, the computational cost of the DNN technique is quite high. In addition, the creation of the network is quite complex as there are many parameters. A distribution similar to this dataset is used in [33]. The result obtained in [33] is 92% according to the accuracy metric. In this study, when a dataset with

TABLE 8. Comparison with previous studies.

Study	Feature Used	Dataset size	Classification Algorithm	Classification Performance
AndroAnalyzer [13]	Permissions	6661 Malware 961 Benign	DNN	0.9820 (f-measure)
Li et al. [11]	Permissions	5494 Malware 5494 Benign	SVM	95.63 (accuracy %)
Milosevic et al. [8]	Permissions	200 Malware 200 Benign	Ensemble learning	0.894 (f-measure)
Milosevic et al. [8]	Source code analysis	200 Malware 200 Benign	Ensemble learning	0.9560 (f-measure)
ANDROIDETECT [9]	System call function	100 Malware 100 Benign	J48	0.86 (f-measure)
Kurniawan et al. [32]	Internet traffic Battery level Battery temperature	200 Malware 200 Benign	RF	85.6 (accuracy %)
MalPat [12]	Permissions API calls	15336 Malware 31185 Benign	RF	0.9824 (f-measure)
Arslan et al. [33]	Permissions	6660 Malware 1074 Benign	Random Tree	92 (accuracy %)
Androdialysis [34]	Permissions Intent	5560 Malware 1846 Benign	Bayesian Network	95.5 (TP rate %)
Sayfullina et al. [35]	13 static features under 4 groups of files	10000 Malware 10000 Benign	Normalized Bernoulli NB	82.10 (TP rate %)
Liu [36]	Permissions	139 Malware 231 Benign	SVM	89.68 (accuracy %)
ALDROID [37]	Permissions Classes.dex file properties	10000 Malware 30000 Benign	SVM	98.8 (accuracy %)
Droidetec [38]	API call sequences	9616 Malware 11982 Benign	Bidirectional long short-term memory (BiLSTM)	97.22 (accuracy %)
Pektaş and Acarman [39]	API call graph	33139 Malware 25000 Benign	Convolutional neural network (CNN)	98.86 (accuracy %)
Our results for AMD	Permissions	1000 Malware 1000 Benign	LinRegDroid Ensemble-1 Ensemble-2	0.956 (f-measure) 0.964 (f-measure) 0.9695 (f-measure)
Our results for Lopez' dataset	Permissions	279 Malware 279 Benign	LinRegDroid Ensemble-1 Ensemble-2	0.9187 (f-measure) 0.9123 (f-measure) 0.925 (f-measure)
Our results for M0Droid	Permissions	200 Malware 200 Benign	LinRegDroid Ensemble-1 Ensemble-2	0.8287 (f-measure) 0.8348 (f-measure) 0.8915 (f-measure)
Our results for Arslan' dataset	Permissions	6661 Malware 961 Benign	LinRegDroid Ensemble-1 Ensemble-2	0.9172 (f-measure) 0.9229 (f-measure) 0.9662 (f-measure)

a similar distribution is used, 98.53% success is achieved with Ensemble-2 according to the accuracy metric. When classification is made with LinRegDroid, 96.69% success is achieved according to the accuracy metric. According to the results obtained from [33], improvement is made between 4% and 6%. In the study conducted by Li *et al.* [11], 95.63% success is obtained according to the accuracy metric. Similar results are obtained using the AMD dataset. When the results of permission-based malware detection systems on small datasets are examined, a performance of 0.894 is obtained according to the f-measure metric in [8]. In [36], an accuracy of 89.68% is obtained according to the accuracy metric. M0Droid dataset is used in [8]. Using this dataset, we achieved 0.8915 performance according to the f-measure metric. Although permission-based approach is used in our study and [8], [36], different structures are presented in classification approaches. However, the results of these three studies are very similar to each other. Lopez's dataset used in this study is also small in size. The performances obtained on this dataset are better than the results obtained from other small datasets since the benign and malware applications can classified more easily in this dataset.

It is observed that performance increases when other attributes such as API calls or intent filters are used together with application permissions [12], [34], [37]. In [35], many static properties are extracted by evaluating 4 different files. However, the performance in [35] is not as high as [12], [34], [37]. When the results of dynamic analysis approaches on small datasets are evaluated, a performance of 0.86 is obtained according to the f-measure metric in [9]. In [32], on the other hand, an accuracy of 85.6% is obtained according to the accuracy metric. When Table 8 is evaluated in general, it is observed that the performance of deep learning techniques is quite good [13], [38], [39]. When the results of the experiments conducted in this study are examined, it is seen that the proposed methods are as successful as the results in the literature.

Remark 2: When the results are examined in general, the researchers generally perform their experiments on the unbalanced dataset. The distribution of the dataset is one of the important factors affecting performance. In the experiments conducted in this study, we usually use a balanced dataset. Another important factor affecting classification performance is feature extraction. Higher classification performances can be achieved as more distinctive features are discovered between benign and malicious applications. These situations differentiate obtained results. For example, experiments are performed using the M0Droid dataset in [8]. Similarly, in this study, experiments are carried out with the M0Droid dataset. The results from both studies are almost

same when extracting permissions from the M0Droid dataset. However, it has been shown that better performance is achieved when the application source codes are used instead of permission [8]. Finally, even if the distributions of the datasets are the same, the characteristics of malware may resemble those of benign. In this case, there may be differences in the performance of classification algorithms.

V. CONCLUSION AND FUTURE WORKS

Application permissions are significant in Android operating system security. These permissions, which are extracted from applications, are used as attributes to detect malicious software with machine learning algorithms in this study. Android malware detection is carried out with two rule-based classification models using multiple linear regression models. The proposed rule-based classifiers are compared with popular classification algorithms such as KNN, NB, SVM, and DT. Both approaches give more successful results than NB and KNN. There are many parameters in SVM, KNN, and NB algorithms. However, classifiers based on multiple linear regression models are quite simple and easy to use. This is the most significant advantage of the proposed approaches. In addition, ensemble learning models based on the bagging technique are also developed in this study. The use of these models positively affects classification performance in general. Finally, in the multiple linear regression model, a large number of models are created by assigning random values to the regression coefficients. However, positive results cannot be obtained from these models. In future studies, it is aimed to create more efficient regression models by developing intelligent search strategies such as hybrid or heuristic techniques.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the anonymous reviewers for their invaluable suggestions in putting the present study into its final form.

REFERENCES

[1] (2021). *Global Market Share Held by the Leading Smartphone Operating Systems in Sales to End Users From 1st Quarter 2009 to 2nd Quarter 2018*. Accessed: Oct. 30, 2021. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[2] (2021). *Malware Disguised as Minecraft Mods on Google Play—Kaspersky Official Blog*. Accessed: Oct. 30, 2021. [Online]. Available: https://www.kaspersky.com/blog/minecraft-mod-adware-google-play-revisited/40202/

[3] H. Wang, Z. Liu, J. Liang, N. Vallina-Rodriguez, Y. Guo, L. Li, J. Tapiador, J. Cao, and G. Xu, "Beyond Google Play: A large-scale comparative study of Chinese Android app markets," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 293–307.

[4] (2021). *Mobile Malware Report—Android Malware*. Accessed: Oct. 30, 2021. [Online]. Available: https://www.gdatasoftware.com/news/2019/07/35228-mobile-malware-report-no-let-up-with-android-malware

[5] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digit. Invest.*, vol. 13, pp. 22–37, Jun. 2015.

[6] A. Pektaş, M. Çavdar, and T. Acarman, "Android malware classification by applying online machine learning," in *Proc. Int. Symp. Comput. Inf. Sci.* Cham, Switzerland: Springer, 2016, pp. 72–80.

[7] M. Dimjašević, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of Android malware detection based on system calls," in *Proc. ACM Int. Workshop Secur. Privacy Analytics*, Mar. 2016, pp. 1–8.

[8] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, "Machine learning aided Android malware classification," *Comput. Electr. Eng.*, vol. 61, pp. 266–274, Jul. 2017.

[9] L. Wei, W. Luo, J. Weng, Y. Zhong, X. Zhang, and Z. Yan, "Machine learning-based malicious application detection of Android," *IEEE Access*, vol. 5, pp. 25591–25601, 2017.

[10] F. Alswaina and K. Elleithy, "Android malware permission-based multi-class classification using extremely randomized trees," *IEEE Access*, vol. 6, pp. 76217–76227, 2018.

[11] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[12] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.

[13] R. S. Arslan, "AndroAnalyzer: Android malicious software detection based on deep learning," *PeerJ Comput. Sci.*, vol. 7, p. e533, May 2021.

[14] D. Ö. Şahin, O. E. Kural, S. Akleylek, and E. Kılıç, "Comparison of regression methods in permission based Android malware detection," in *Proc. 28th Signal Process. Commun. Appl. Conf. (SIU)*, Oct. 2020, pp. 1–4.

[15] Ö. Polat, "A robust regression based classifier with determination of optimal feature set," *J. Appl. Res. Technol.*, vol. 13, no. 4, pp. 443–446, Aug. 2015.

[16] M. Khashei, M. Bijari, and A. Z. Hamadani, "A novel hybrid classification model of artificial neural networks and multiple linear regression models," *Expert Syst. Appl.*, vol. 39, no. 3, pp. 2606–2620, 2012.

[17] L. Tang, H. Lu, Z. Pang, Z. Li, and J. Su, "A distance weighted linear regression classifier based on optimized distance calculating approach for face recognition," *Multimedia Tools Appl.*, vol. 78, no. 22, pp. 32485–32501, Nov. 2019.

[18] H. Wang and F. Hao, "An efficient linear regression classifier," in *Proc. IEEE Int. Conf. Signal Process., Comput. Control*, Mar. 2012, pp. 1–6.

[19] I. Naseem, R. Togneri, and M. Bennamoun, "Linear regression for face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 11, pp. 2106–2112, Nov. 2010.

[20] A. Seal, D. Bhattacharjee, M. Nasipuri, and D. K. Basu, "UGC-JU face database and its benchmarking using linear regression classifier," *Multimedia Tools Appl.*, vol. 74, no. 9, pp. 2913–2937, May 2015.

[21] R. Jusoh, A. Firdaus, S. Anwar, M. Z. Osman, M. F. Darmawan, and M. F. A. Razak, "Malware detection using static analysis in Android: A review of FeCO (features, classification, and obfuscation)," *PeerJ Comput. Sci.*, vol. 7, p. e522, Jun. 2021.

[22] V. Kouliaridis and G. Kambourakis, "A comprehensive survey on machine learning techniques for Android malware detection," *Information*, vol. 12, no. 5, p. 185, Apr. 2021.

[23] J. Senanayake, H. Kalutarage, and M. O. Al-Kadri, "Android mobile malware detection using machine learning: A systematic review," *Electronics*, vol. 10, no. 13, p. 1606, Jul. 2021.

[24] (2021). *AAPT2 | Android Developers*. Accessed: Oct. 30, 2021. [Online]. Available: https://developer.android.com/studio/command-line/aapt2

[25] T. G. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 40, no. 2, pp. 139–157, 2000.

[26] M. Damshenas, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, "M0Droid: An Android behavioral-based malware detection model," *J. Inf. Privacy Secur.*, vol. 11, no. 3, pp. 141–157, Sep. 2015.

[27] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 252–276.

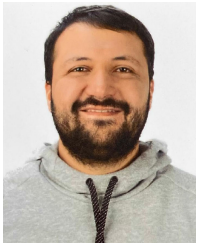
[28] (2019). *Android Malware Dataset*. Accessed: Oct. 15, 2019. [Online]. Available: http://amd.arguslab.org/

[29] (2021). *APKPure Android Application Store*. Accessed: Oct. 30, 2021. [Online]. Available: https://apkpure.com

[30] C. Urcuqui-López and A. N. Cadavid, "Framework for malware analysis in Android," *Sistemas Y Telemática*, vol. 14, no. 37, pp. 45–56, 2016.

[31] (2021). *Access to Dataset*. Accessed: Oct. 30, 2021. [Online]. Available: https://kaggle.com/xwolf12/datasetandroidpermissions

[32] H. Kurniawan, Y. Rosmansyah, and B. Dabarsyah, "Android anomaly detection system using machine learning classification," in *Proc. Int. Conf. Electr. Eng. Informat. (ICEEI)*, Aug. 2015, pp. 288–293.
[33] R. S. Arslan, İ. A. Doğru, and N. Barişçi, "Permission-based malware detection system for Android using machine learning techniques," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 29, no. 1, pp. 43–61, Jan. 2019.
[34] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: Analysis of Android intent effectiveness in malware detection," *Comput. Secur.*, vol. 65, pp. 121–134, Mar. 2017.
[35] L. Sayfullina, E. Eirola, D. Komashinsky, P. Palumbo, Y. Miche, A. Lendasse, and J. Karhunen, "Efficient detection of zero-day Android malware using normalized Bernoulli Naive Bayes," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 198–205.
[36] W. Liu, "Mutiple classifier system based Android malware detection," in *Proc. Int. Conf. Mach. Learn. Cybern.*, vol. 1, Jul. 2013, pp. 57–62.
[37] N. Nissim, R. Moskovitch, O. Barad, L. Rokach, and Y. Elovici, "ALDROID: Efficient update of Android anti-virus software using designated active learning methods," *Knowl. Inf. Syst.*, vol. 49, no. 3, pp. 795–833, Dec. 2016.
[38] Z. Ma, H. Ge, Z. Wang, Y. Liu, and X. Liu, "Droidetec: Android malware detection and malicious code localization through deep learning," 2020, *arXiv:2002.03594*.
[39] A. Pektaş and T. Acarman, "Deep learning for effective Android malware detection using API call graph embeddings," *Soft Comput.*, vol. 24, no. 2, pp. 1027–1043, Jan. 2020.



DURMUŞ ÖZKAN ŞAHİN received the B.Sc. degree in computer engineering from Süleyman Demirel University, Isparta, in 2013, and the M.Sc. degree in computer engineering from Ondokuz Mayıs University, Samsun, in 2016, where he is currently pursuing the Ph.D. degree in computational sciences. His research interests include machine learning, text mining, information retrieval, and android malware analysis.

SEDAT AKLEYLEK received the B.Sc. degree in mathematics majored in computer science from Ege University, Izmir, Turkey, in 2004, and the M.Sc. and Ph.D. degrees in cryptography from Middle East Technical University, Ankara, Turkey, in 2008 and 2010, respectively. He was a Postdoctoral Researcher at the Cryptography and Computer Algebra Group, TU Darmstadt, Germany, between 2014 and 2015. He has been an Associate Professor at the Department of Computer Engineering, Ondokuz Mayıs University, Samsun, Turkey, since 2016. His research interests include the areas of post-quantum cryptography, algorithms and complexity, architectures for computations in finite fields, applied cryptography for cyber security, malware analysis, the IoT, and fog computing. He is a member of the Editorial Board of IEEE Access, *Turkish Journal of Electrical Engineering and Computer Sciences*, *Peerj Computer Science*, and *International Journal of Information Security Science*.

ERDAL KILIÇ received the B.Sc. degree in electrical electronic engineering and the M.Sc. degree in electrical electronic engineering from Karadeniz Technical University, Trabzon, in 1991 and 1996, respectively, and the Ph.D. degree in electrical and electronic engineering from Middle East Technical University, Ankara, in 2005. Currently, he is a Full Professor at the Department of Computer Engineering, Ondokuz Mayıs University. His research interests include neural networks, machine learning, and data mining.