Hindawi
Security and Communication Networks
Volume 2021, Article ID 6642252, 13 pages
https://doi.org/10.1155/2021/6642252

WILEY | Hindawi

Research Article

FB2Droid: A Novel Malware Family-Based Bagging Algorithm for Android Malware Detection

Ke Shao,[1,2] **Qiang Xiong**,[3] **and Zhiming Cai**[1]

[1]*City University Macau, Institute of Data Science, Macau 999078, China*
[2]*School of Computer Technology, Beijing Institute of Technology Zhuhai, Zhuhai 519088, China*
[3]*Zhongkai University of Agriculture and Engineering, Guangzhou 510225, China*

Correspondence should be addressed to Zhiming Cai; caizhiming@cityu.mo

Received 29 December 2020; Revised 23 April 2021; Accepted 7 June 2021; Published 21 June 2021

Academic Editor: Jesús Díaz-Verdejo

Copyright © 2021 Ke Shao et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As the number of Android malware applications continues to grow at a high rate, detecting malware to protect the system security and user privacy is becoming increasingly urgent. Each malware application belongs to a specific family, and there is a gap in the number of malware families. The accuracy of detection can be improved if malware family information is well utilized and certain strategies are adopted to balance the variability among samples. In addition, the performance of a base classifier is limited. If an ensemble classifier or an ensemble method can be adopted, the detection effect can be further improved. Therefore, this paper proposes a novel malware family-based bagging algorithm for Android malware detection, called FB2Droid, to perform malware detection. First, five features are extracted from the Android application package. Then, the relief feature selection algorithm is used for feature selection. Next, we designed two different sampling strategies based on different families of malware to alleviate the sample imbalance in the dataset. Combined with the two sampling strategies, the traditional bagging algorithm is improved to integrate the classifier. In the experiment, several classifiers were used to evaluate the proposed scheme. The experimental results show that the proposed sampling strategy and the improved bagging algorithm can effectively improve the detection accuracy of these classifiers.

1. Introduction

In recent years, the Android operating system has developed rapidly and has been the most widely used smartphone operating system. According to a recent report in the Global Stats [1], as of April 2019, Android devices have the highest market share among all smartphone users, accounting for 74.85%. As well as its popularity, it has caught the attention of malware creators. Regarding Android malware and based on the McAfee Mobile Threat Report 2020, the total number of mobile malware detected was more than 35 million in Q4 of 2019 [2]. The phenomenal growth of malware poses a great threat to users; therefore, it is urgent to detect malware and protect user security.

With the development and popularity of machine learning algorithms, they are widely used in the field of Android malware detection. Machine learning-based malware detection mainly includes the following steps: (1) data collection and collation, and analysis of benign and malicious samples to extract features; (2) use the feature selection algorithm to screen extracted features; (3) training classification model; (4) using classification models to detect and judge unknown samples. According to different feature extraction methods, Android malware detection can be divided into static analysis and dynamic analysis. Static analysis uses reverse engineering to decompile the Android application package (APK) without installing and executing the application and then extracts features from the code and other files. Unlike static analysis, dynamic analysis requires running an application and then gathering information about what is running to extract features. This method can achieve high accuracy in identifying malicious activities, but it takes a long time and huge resources to run the monitor in real time. In contrast, static analysis technology is widely used because of its low resource consumption and high code coverage.

The problem of Android malware detection is essentially a classification problem. With the rapid development of malware, there are various types of malware and a large number of them. According to the behavior and characteristics of malware, malware can be divided into different families. However, the development of different malware families is uneven, and the number of samples varies greatly, which leads to the imbalance in the detection. In other words, the large number of samples of a certain type leads to the "bias" of the classifier towards such samples, thus affecting the detection effect. Traditional classification algorithms have achieved good results in the balanced datasets, but the actual datasets are often unbalanced, and the traditional algorithms are sensitive to unbalanced data and have poor detection effects, such as SVM [3, 4] and RF [5, 6]. The data imbalance [7, 8] is manifested in two aspects: (1) interclass imbalance, where the number of samples in one class in the dataset is significantly less than the number of samples in other classes; (2) intraclass imbalance, where the number of samples in a class and its subclasses is unbalanced or a class of data presents multiple small separated terms. Since traditional classifiers take the overall classification accuracy as the learning goal, in this case, to obtain a greater classification accuracy, it is bound to cause the classifier to pay too much attention to the majority of the class samples, thus reducing the classification performance of the minority of the class samples. At present, most malware detection schemes ignore the intraclass imbalance, that is, the imbalance between various software families within the malware, which leads to the trained classifier having better detection performance on the families with large sample size, but less performance on the families with the small number of samples, thus reducing the overall detection efficiency. Besides, most of the existing Android malware detection schemes use only a single classifier with limited capability and effectiveness.

Therefore, to address the above issues, this paper proposes a novel malware family-based bagging algorithm for Android malware detection. First, the scheme extracts five features from APK files, then uses the relief feature selection algorithm to evaluate the importance of each feature, and then selects some of the more important features for classifier training and malware detection. Then, based on the family tags provided in the dataset, two sampling strategies are designed to alleviate the intraclass imbalance in the dataset. Then, the designed strategy is used to improve the bagging algorithm to integrate classifiers to detect malware. The experimental results show that the detection accuracy and F-score of malware are improved by 2.3% and 2.0% by using the proposed ensemble scheme based on some common classifiers.

The contributions of this paper are as follows:

(1) The family information and category labels of malware are applied to malware detection, and an integrated scheme of bagging based on malware families is constructed to effectively improve the accuracy of malware detection.

(2) Considering the imbalance between malware families, the sampling strategy of bagging is improved. On the basis of random sampling based on the classical bagging algorithm, two sampling methods are added: the equal number sampling strategy based on the number of malware families and the sampling strategy based on the malware family information.

(3) The designed sampling strategies take into account the imbalance between the malware families, can effectively alleviate the imbalance within the class, and fully take care of each malicious family

(4) We designed and carried out a number of experiments and presented a wealth of experimental results. The experimental results show the feasibility and effectiveness of the proposed scheme.

The rest of this paper is organized as follows. The related work is introduced in Section 2. The feature extraction and selection methods are described in Section 3. Section 4 describes the proposed bagging ensemble scheme and designed sampling strategies. Section 5 presents the evaluation and analysis. Finally, conclusions are drawn in Section 6.

2. Related Work

Android malware detection technologies fall into three categories: static analysis, dynamic analysis, and hybrid analysis. Static detection uses corresponding decompilation tools to extract the static features of the program, such as syntax-semantics and signatures, for analysis, to evaluate application security and detect malware. And, static analysis can be further subdivided. One is signature-based malware detection. The signature-based static analysis method mainly uses the unique signature mechanism of the application software to uniquely identify the signature and feature code of the application and compare it with the known malware feature base, and if there is a match to the consistent signature, it is judged as malware. Faruki et al. [9] presented AndroSimilar, an approach that generates signatures by extracting statistically robust features, to detect malicious Android apps. Zheng et al. [10] proposed DroidAnalytics, a signature-based detection and analysis system, that automatically collects, manages, analyzes, and extracts Android malware. It uses a multilevel signature algorithm (signatures contain method level, class level, and application level) to extract malware features based on opcode level semantics. The multilevel signature approach offers important advantages over traditional encryption methods (e.g., md5-based signatures). However, this technique has some limitations and is not effective in solving the problem with high accuracy when used to detect repackaged malware. And, with the explosive growth of malicious applications, new malicious application feature signature codes need to be extracted continuously, which increases the workload to some extent and is too inefficient. This gives rise to an alternative static analysis method that uses multiple static

features to detect malware. For this purpose, the executable application (APK) is reverse engineered from which relevant features are extracted for analysis and detection, such as API [11–13], permission [14, 15], intents [16], function call graph [17, 18], and control follow [19]. Arora et al. [20] proposed a novel detection model, called PermPair, which detects malware by mining the combination of permissions from the AndroidManifest.xml on applications. Niu et al. [21] used function call to construct function call graphs to classify Android malware. OpCode-level FCG was used in this paper, which is obtained through static analysis of Operation Code (OpCode). Then, authors applied Long Short-Term Memory (LSTM) to detect applications.

Different from static analysis, dynamic analysis monitors the operation of the program in real time to find and detect malicious behavior of the program. Among them, the dynamic feature includes network traffic [22, 23], system calls [24–26], and resource consumption [27]. Vinod et al. [25] proposed the Android malware detection scheme based on system calls, and the results are verified on five datasets. RoughDroid [28] proposed a floppy analysis technique that can discover Android malware applications directly on the smartphone. However, dynamic analysis needs to monitor applications on a mobile device or simulator to extract features, which is time-consuming. Therefore, some scholars have combined static analysis and dynamic analysis to produce hybrid analysis. In [29], the authors proposed a hybrid detection method that performs dynamic detection on these suspicious applications which are detected by static detection. In [30], the authors compiled static and dynamic features from benign and malicious applications such as API call sequence, system command, manifest permission, and intent. Then, they used a deep neural network to classify applications. In [31], the authors applied network-related features and activity bigrams to construct a detection scheme to resolve the Android malware familial classification problem, which is based on machine learning and hybrid analysis. An extensive feature set including network-related features and activity bigrams is proposed. The proposed scheme in this paper belongs to static analysis.

With the rapid development of Android malware, there are a variety of different types of malware, which can be divided into different families according to the characteristics of malware. The classification of the malware family is also an important part of malware detection. However, there are many kinds of malware families in a dataset, and the distribution among them is unbalanced, which makes it difficult to detect the malware. Therefore, many scholars have begun to focus on this direction, hoping to improve the detection efficiency through the classification of the malware family. EC2 [32] presented a novel algorithm for discovering Android malware families of varying sizes—ranging from very large to very small families (even if previously unseen). Fan et al. [33] proposed a new method to construct fregraphs to represent the common behaviors of malware samples belonging to the same family, classify Android malware, and select representative malware samples according to fregraph. Then, they designed and implemented a new detection system, FalDroid, which can handle the family classification of large-scale Android malware with high precision and efficiency. This paper only used the family information of malware to improve the bagging algorithm to detect malware.

Due to the development of machine learning, it has been widely used in Android malware detection. However, it is difficult to improve the detection results by using a single base classifier, such as support vector machine (SVM) [34, 35]; therefore, it has become a trend to use ensemble classifiers for detection. The way to combine classifiers that have independent decision-making ability with each other is called an ensemble classifier. It turns out that the predictive ability of integrated classifiers is much better than that of individual classifiers in general. In the studies using the ensemble classifier, there are two directions: one is to directly use related ensemble classifiers, such as random forest (RF) [36], gradient boosting decision tree (GBDT) [37], and AdaBoost [38]; the other is to use ensemble methods to combine with single classifiers, such as boosting, bagging, and stacking methods. Yerima and Sezer [39] proposed a detection framework, DroidFusion, based on multilevel classifier fusion. This framework uses a sorting algorithm to sort the low-level training classification model and select the optimal combination to produce the final decision result. In addition, in the field of malware detection, machine learning algorithms have significant advantages in feature extraction, optimization, and software detection. Therefore, this paper chooses to improve the bagging algorithm to further enhance the detection performance of multiple machine learning.

3. Feature Extraction and Selection

3.1. Feature Extraction. The dataset of Android malware detection is the original APK file, which is the installation package of Android applications. In order to extract the relevant features, the APK file needs to be decompiled to obtain the program code and its key files. Therefore, in this paper, the APK is decompiled using the Androguard [40] tool to obtain the AndroidManifest.xml and Class.dex file and convert the Class.dex file into a .smali file which is more human-friendly as intermediate presentations of bytecode. The AndroidManifest file contains all the permissions required for the execution of the program and communication components. Therefore, in this paper, the following five features are extracted from the above two documents, as shown in Figure 1:

(i) Permission: if an app needs to execute some specified operations, it must request corresponding permissions

(ii) App components: application components are the essential building blocks of an application including activities, services, providers, and receivers

(iii) Intent filters: intents handle the communication between components

(iv) Sensitive API calls: some API calls allow access to sensitive data and resources on smartphones

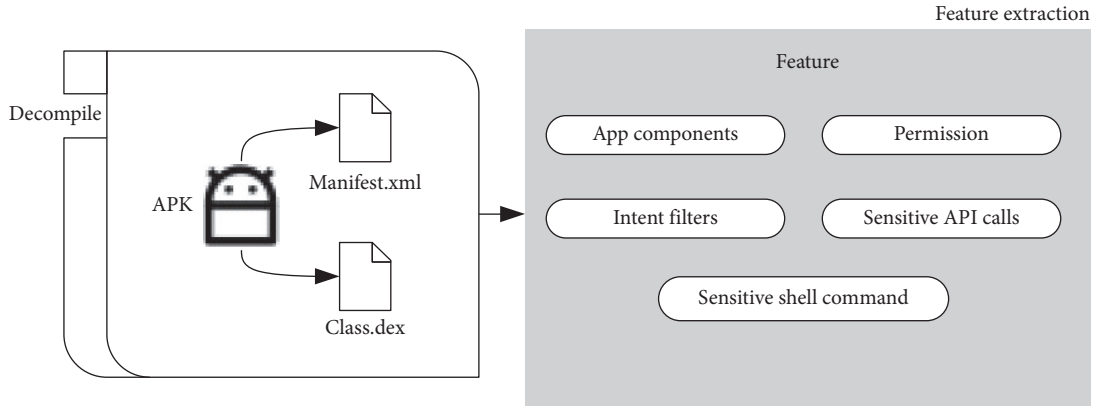

Figure 1: Feature extraction flow chart.

(v) Sensitive shell command: some commands can gain root privilege and perform dangerous operations

In the experiment, to reduce resource consumption and speed up detection efficiency, the feature is represented in binary form. If the feature exists in the application, the corresponding location of the feature is marked as 1. Otherwise, it is marked as 0 and an eigenmatrix of 0/1 is finally formed as shown in Figure 2.

To further explain the eigenmatrix, assume that the number of samples is m and the number of features is n. Thus, the two-dimensional eigenmatrix of Figure 2 is obtained, where each row represents a sample and each column represents a feature.

	Feature_1	...	Feature_n
APK_1	1	...	0
⋮	⋮	⋱	⋮
APK_m	0	...	1

Figure 2: Feature representation.

3.2. Feature Selections. The data and feature determine the upper bound of machine learning, and the model and algorithm only approximate the upper bound. Feature selection is to extract features from the original data to the greatest extent for the use of algorithms and models. Generally, feature selection consists of the following four steps:

(i) Generation: generate the candidate subset of features

(ii) Evaluation: evaluate the quality of the feature subset

(iii) Termination: decide when to stop the generation procedure

(iv) Validation: check whether the feature subset is valid

Feature selection techniques aim to gain a deeper understanding of the data and to minimize the amount of computation required during the training and testing phases to improve prediction performance. It also generalizes the learning model to reduce overfitting. Generally, feature selection algorithms are divided into two methods: (a) feature search and (b) feature subset evaluation. Feature search is one of the techniques widely used in attribute space research. In the field of machine learning, feature search can be divided into the forward selection and backward elimination. Feature subset evaluation is an approach that is widely used to identify irrelevant and redundant attributes. There are three types of feature assessment techniques: filtering, which rates features according to divergence or correlation and sets thresholds to select features; wrapper, which selects or excludes features each time according to the objective function (usually the predictive effect score); embedding is trained by using the machine learning algorithm and model, and weight coefficients of various features are obtained. Features are selected according to the coefficients ranging from large to small. In contrast, filter has high computational efficiency, is independent of other classification learning methods, and can effectively rank the importance of features. Therefore, this paper selects the relief algorithm for feature selection.

As an effective feature selection algorithm, the relief [41] algorithm assigns different weights to features according to the correlation of each feature and category, and features with weights smaller than a certain threshold value will be removed. The relief algorithm randomly selects a sample R from the training set D, then finds the nearest neighbor sample H from the samples of the same category as R, which is called Near Hit, finds the nearest neighbor sample $M(C)$ from the samples of different kind of R, which is called Near Miss, and then updates the weight of each feature according to the following rules:

$$W(F) = W(F) - \frac{\text{diff}(F, R, H)}{m} + \frac{\text{diff}(F, R, M(C))}{m}, \quad (1)$$

where F denotes a feature in the sample, m is the number of samples, $\text{diff}(F, R_1, R_2)$ function represents the difference between sample R_1 and sample R_2 on feature F, and $M(C)$ represents the nearest neighbor sample in category C.

If the distance between R and Near Hit on a feature is less than the distance between R and Near Miss, it indicates that the feature is beneficial to distinguish the nearest neighbor of a similar species from a different one; then, the weight of the feature will be increased. On the contrary, if the distance

between R and Near Hit on a feature is greater than the distance between R and Near Miss, it indicates that this feature has a negative effect on distinguishing the nearest neighbors of similar species and different species; then, the weight of this feature will be reduced. The above process is repeated m times, and finally, the average weight of each feature is obtained. Therefore, according to the weight calculated by the relief algorithm, the feature of small weight was filtered out in this paper, which effectively improved the detection efficiency. See Section 5 for a detailed description.

4. Ensemble Detection Scheme

Traditional Android malware detection schemes based on machine learning usually only use a single base classifier such as SVM for detection, with limited performance. However, better results can be achieved by combining multiple base classifiers or using ensemble classifiers.

4.1. Bagging Algorithm. The bagging algorithm was the most famous representative of the parallel ensemble algorithm. In the ensemble algorithm, the bagging method is to build multiple instances of a class of black box evaluators on a random subset of the original training set and then combine the prediction results of these estimators to form the final prediction result. Idea of the algorithm is to make learning algorithm training multiple rounds; each round of the training set consists of M samples randomly taken from the original training set, and a training sample can appear multiple times in one round of training or did not appear (that is, sampling with replacement). A prediction function can be obtained after training; finally, classification problem is solved by voting. The famous integrated classifier RF, the bagging algorithm plus decision tree (DT) classifier, was proposed by Breiman [42]. The basic flow of the bagging algorithm, as shown in Algorithm 1, includes sampling, training, and classification. Therefore, the improved bagging integration scheme proposed in this paper is an improvement and innovation in sampling and final voting classification.

4.2. Sampling Strategy. The traditional bagging integration algorithm uses a random sampling strategy where each sample is selected with the same probability, but in reality the data is unbalanced and the variability between different malware families is sometimes large, i.e., the number of samples varies widely with the size. In this case, the distribution of samples in the sampling set applying random sampling is extremely unbalanced, with some families having a large number and others having very few, resulting in a low recognition rate for some of the family samples thus reducing the overall detection performance. However, the traditional bagging algorithm only uses a random sampling strategy with limited performance improvement. Secondly, considering that each sample in the dataset belongs to a malware family and the number of families varies, it is difficult for the traditional bagging algorithm to take into account each malicious family. Therefore, in order to make full use of the family information of each malware and improve the detection performance, two sampling strategies are added in this paper based on the traditional bagging algorithm, as shown in Figure 3: the equal quantity sampling strategy based on the malware family and the family sampling strategy based on the malware family information. Sampling by equal number of families means that each family in the training set takes benign and malicious samples by the same number, and those malicious families with insufficient samples make up the number of samples to be sampled by oversampling.

The family sampling with the malware family information is based on the calculated family information of each sample. Each malware belongs to a specific malware family, and each malware contains information about its family. Before starting sampling, a family information estimator is trained for each malware family in the training set to compute the family information of each sample. All the malicious samples of each family and all the benign samples in the training set are sent to the family information estimator for training, and then, the family information contained in each sample is output. Finally, the number of samples that each family should take is calculated according to the family information of all the samples in each malicious family. Among them, the malware family information MFI of a sample represents the probability that the sample belongs to the family, and the higher the MFI value, the more likely it is to belong to the family. The MFI calculation formula is as follows:

$$\mathrm{MFI} = \mathrm{Estimator}(\mathrm{sample}), \tag{2}$$

where the Estimator() denotes a family information estimator.

On the contrary, the probability of a malicious sample being misjudged as a benign application is BFI. The BFI calculation formula is as follows:

$$\mathrm{BFI} = 1 - \mathrm{MFI}. \tag{3}$$

Therefore, the greater the sum of the probability of all samples being misjudged in a certain family, the more difficult it is to detect the malicious family, and more samples are needed to participate in classifier training. Therefore, this paper designs the following formula to calculate the number of samples to be taken:

$$N_i = \left[\mathrm{Num}\left(\mathrm{family}_i\right) * \sum_{j=1}^{\mathrm{Num}\left(\mathrm{family}_i\right)} \mathrm{BFI}_{i,j} \right], \tag{4}$$

where N_i denotes the number of samples to be sampled from the family i, $\mathrm{BFI}_{i,j}$ denotes the probability that sample j in the family i being misjudged as a benign application, and $\mathrm{Num}(\mathrm{family}_i)$ represents the number of samples in malware family i.

Both of these new sampling strategies are based on malware families, and the number of families determines the number of sampling. In the subsequent experiments, to reduce the complexity of the scheme, the choice of family information estimators in the family sampling strategy based

Input: Dataset D, number of iterations T, number of sampling n, base classifier $G(x)$
Output: The final strong classifier $f(x)$
(1) for $t = 1, 2, \ldots, T$;
(2) The training set D is randomly sampled for the tth time, and a total of m times are taken to obtain a sampling subset D_t containing m samples.
(3) Training the base classifier $G_t(x)$ using the sample set D_t.
(4) If it is a classification algorithm, the voting strategy is adopted, that is, the category or one of the categories with the most votes cast by T base classifiers is the final classification. If it is a regression algorithm, the arithmetic mean is adopted, i.e., the arithmetic mean is performed on the regression results obtained by T weak classifiers, and the value obtained is the final model output.

Algorithm 1: The basic flow of the bagging algorithm.

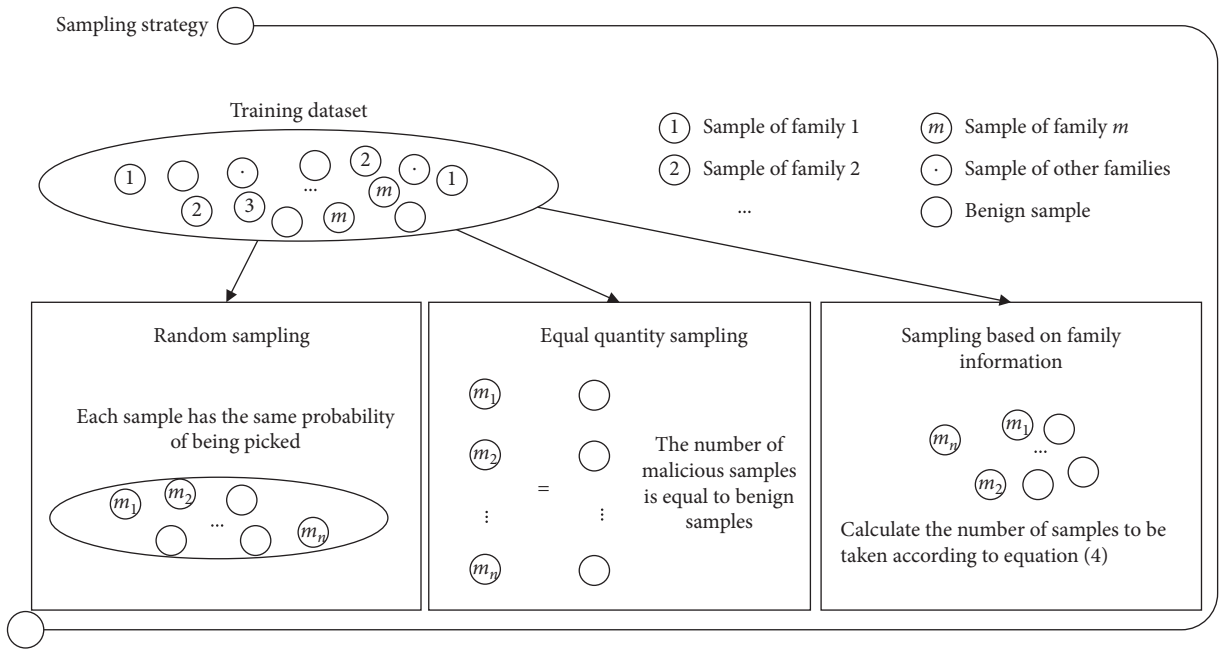

Figure 3: Sampling strategy.

on malware family information is not discussed anymore, and the information estimators of each family adopt the same classification model, and the classification model depends on the single classifier to be integrated by the ensemble scheme.

4.3. Improved Bagging Ensemble Detection Scheme. In the ensemble detection scheme proposed in this paper, each round of sampling will train multiple base classifiers, and the difference of base classifiers can effectively improve the detection performance of the bagging method. Therefore, to further improve the difference between base classifiers, two sampling strategies are added in this paper based on the traditional bagging algorithm. Therefore, this scheme uses three sampling strategies, respectively, and each sampling strategy is executed N rounds. In each round, n samples are selected from the original training set as training subsets, and a total of 3N training subsets are selected. A base classifier is trained on each subset, and each base classifier remains independent of the others. The three sampling strategies can be executed in parallel.

First, we perform sampling in the divided training set. The random sampling strategy, the equal number sampling strategy based on malware families, and the family sampling strategy based on malware family information are used, respectively, and there are put-back N rounds of sampling, and each sampling method gets N training subsets D_i, $i = 1, 2, \ldots, n$, for a total of $3N$ training subsets. Next, a base classifier is trained for each training subset T_i. When the family sampling strategy based on malware family information is adopted, family information estimators should be trained separately for each family to calculate the number of samples. Finally, the base classifiers are trained by drawing samples according to the calculated number of samples, and then, the base classifiers are combined into a strong classifier using a weighted voting strategy with optimal weights so that the samples in the test set, one by one, pass through each base classifier and perform weighted voting to output the final classification results. In the final weighted voting, since three different sampling strategies are used, the detection performance of the trained base classifiers are different, and each classifier cannot be combined by simply using the

voting strategy, and appropriate weights are assigned to each classifier to further highlight the differences between classifiers to improve the detection effects. In order to assign optimal weights to each base classifier, this paper uses an intelligent optimization algorithm to automatically optimize the weights of each classifier. Therefore, this paper uses a differential evolution (DE) algorithm for adaptive optimization. The DE algorithm is a random search global optimization algorithm based on population evolution. It uses real number coding and includes three basic operations: mutation, crossover, and selection. The optimization search is guided by heuristic population intelligence generated by mimicking cooperation and competition among biological populations. Compared with other evolutionary algorithms, the DE algorithm has the advantages of fast convergence, simple operation, and easy implementation. When using the DE algorithm, based on experience value and experimental verification, the population is set to 40, the number of evolution is set to 30, the scaling factor is 0.5, crossover probability is 0.3, and the classifier's accuracy as the fitness function, and in order to have more stability and authenticity in the experimental results, a five-fold crossover validation is used. Also, when the DE algorithm is used to optimize the weight coefficient of the base classifier, in order to improve the learning efficiency of the model, reduce the amount of calculation, and at the same time to ensure the difference between the base classifiers as far as possible; the basis classifiers obtained by the same sampling algorithm have the same weight.

The rule of weighted voting strategy is to multiply the predicted votes of each base classifier by their weight coefficients and then sum up the weighted votes of each category, and the corresponding category of the obtained value is the final classification result. The weighted voting formula is as follows:

$$R(x_i) = \operatorname{sign}\left(\sum_{j=1}^{3N} w_j * C_j(x_i)\right), \tag{5}$$

where w_j denotes the weight of each classifier and $C_j(x_i)$ denotes the prediction result of the sample on classifier C_j. The overall flow of the detection scheme is shown in Figure 4.

The basic process of the ensemble detection scheme is as follows:

(1) The original dataset is divided into a training dataset and testing dataset
(2) For a training dataset, three sampling strategies are used to carry out N replacements sampling, and each sampling method obtains N training subsets, respectively, with a total of $3 \cdot N$ subsets
(3) A base classifier is trained for each training subset, and the base classifiers are independent of each other
(4) Differential evolution algorithm was used to optimize the weight of the base classifiers
(5) Let each sample in the testing dataset be detected by each base classifier, and the results are weighted and voted on to output the final classification result for each sample

5. Evaluation

To verify the scheme presented in this paper, the following experiments are performed. First, we show the distribution of the malicious family samples in the training set. Then, the features optimized by the feature selection algorithm are discussed. Next, it is verified whether the optimal weights of the weighted voting strategy have an improvement in the performance. Finally, the proposed scheme is compared with traditional classification methods such as SVM, RF, DT, *K*-Nearest Neighbor (KNN), Logistics Regression (LR), Linear Discriminant Analysis (LDA), Multilayer Perceptron (MLP), RF, AdaBoost, and GBDT.

5.1. Datasets and Selected Feature. The dataset used in the experiment in this paper includes malicious samples and benign samples. The malicious samples are mainly derived from the AMD [43] dataset, which contains 71 families with a total of 24,553 malware samples and is widely used in the field of Android malware detection. Benign samples come from Google Play Store [44] and APKPure [45]. At the same time, to ensure the effectiveness of the experiment, the application of the benign dataset includes sports, business, games, education, news, and other categories, taking into full consideration the multiple types of data samples in Android applications.

The next experiment randomly selected 8000 malicious samples from the AMD dataset and 8000 benign samples from the benign dataset at the same time, totaling 16000, to form the experimental dataset and divided into training and test sets in the ratio of 7 : 3. When sampling, samples from the original training set were taken as the training set, and 5 rounds of each sampling method were taken, respectively. Figure 5 shows distribution of the top 10 malicious families in the divided training set. As can be seen, the sample distribution among the malicious families is extremely uneven, and the gap between the number of the first family and the tenth family has reached about ten times, not to mention the small number of malicious families. Therefore, it is extremely important to mitigate the intraclass imbalance of malicious samples.

In addition, the features used in the experiment were filtered, and the weight of each feature was calculated by using the relief algorithm and ranked from large to small. Then, features with a weight greater than 0.003 were selected as the final input features. The number of features decreased from 89756 to 789, greatly reducing feature redundancy and improving detection efficiency.

Malware detection is a classification problem. Four common metrics exist for classification problems: accuracy, precision, recall rate, and *F*-score. The malicious sample was denoted as the positive (P) class, and the benign sample was denoted as the negative (N) class. Then, four numbers are obtained:

(i) True positive (TP): the number of positive samples that are correctly predicted as positive
(ii) False negative (FN): the number of positive samples that are incorrectly predicted as negative

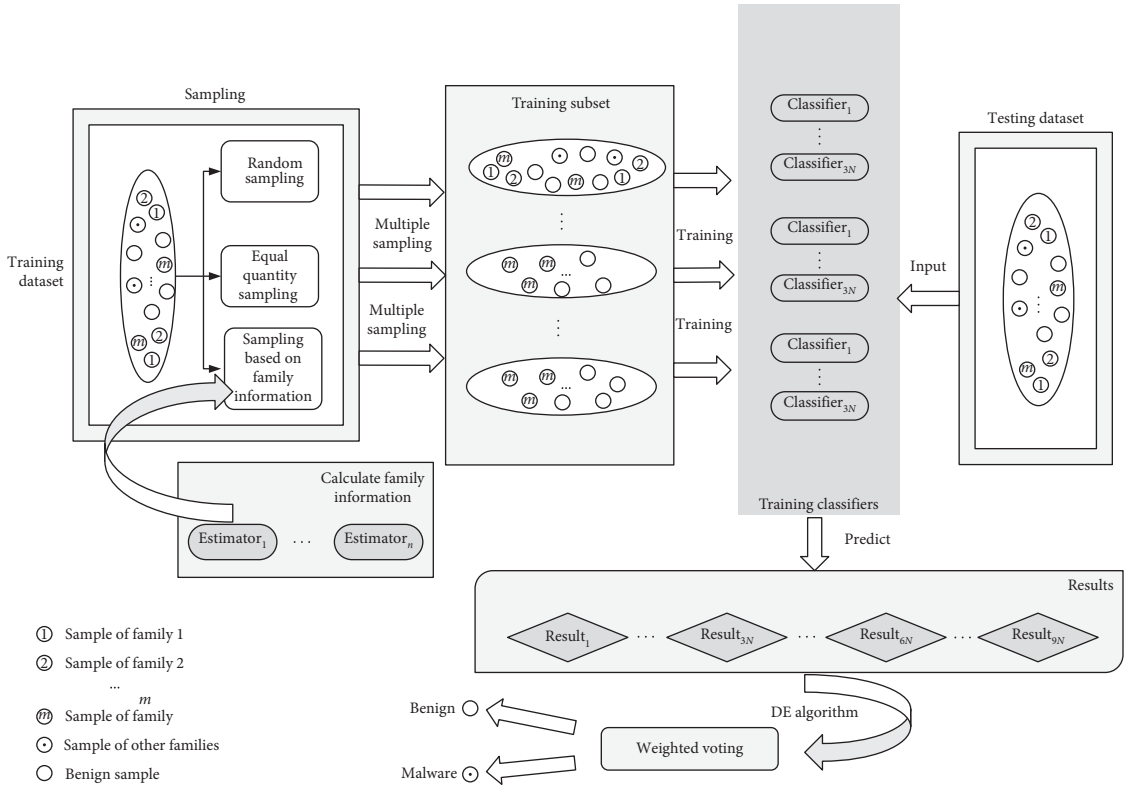

Figure 4: Ensemble detection scheme.

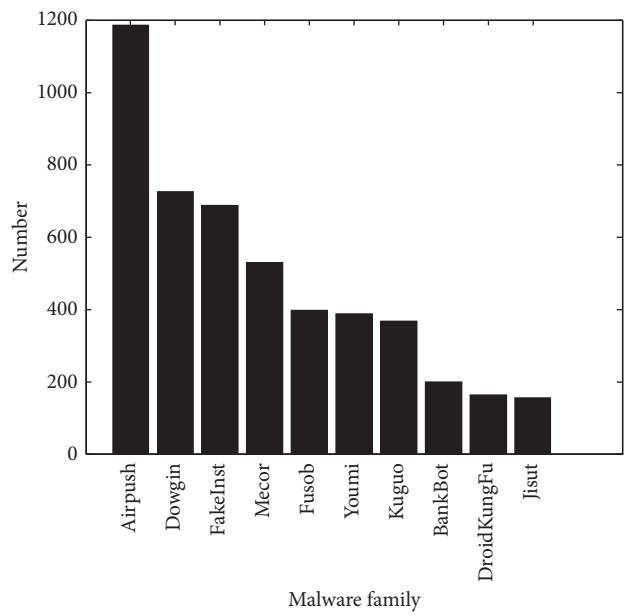

Figure 5: The top 10 family number distribution.

(iii) False positive (FP): the number of negative samples that are incorrectly predicted as positive

(iv) True negative (TN): the number of negative samples that are correctly predicted as negative

Based on these designations, the following equations calculate four common metrics:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}, \tag{6}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{8}$$

$$F - \text{score} = \frac{2}{(1/\text{precision}) + (1/\text{recall})} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}. \tag{9}$$

The precision and recall rate affect each other, and their harmonic mean is the F-score.

All experiments were run on a personal computer with an Intel (R) Core (TM) i7-9700 @ 3.0 GHz CPU with 32 GB of memory and a Windows 10 64-bit operating system. The software used for evaluation includes Python 3.6.1 and sklearn 0.23.2.

5.2. Performance of New Sampling Strategy. The FB2 ensemble scheme proposed in this paper designs two new malicious family-based sampling strategies: the equal number sampling strategy based on malware families, and the family sampling strategy based on malware family information. In order to verify the performance of the two sampling strategies, we compare them with the traditional bagging integration algorithm without the weighted voting

strategy (each classifier weight is set to 1, which is consistent with the traditional bagging integration algorithm), and the results are shown in Table 1.

The results show that the traditional bagging algorithm has achieved good performance without using weight, but the performance of the FB2 ensemble scheme has been further improved. Therefore, the two sampling strategies based on malicious families designed in this paper are helpful to alleviate the imbalance within the malicious samples and improve the detection performance.

5.3. Performance of the Optimal Vote Weight. The bagging ensemble scheme proposed in this paper not only designs two new sampling strategies but also improves the combination of each final base classifier, i.e., the weighted voting strategy is adopted to carry out the final integration. Due to the different sampling strategies and the different samples selected in each round of sampling, the base classifiers finally trained are also different, so each base classifier has its own preference or difference. The more obvious the difference between base classifiers, the better the final detection. Therefore, this paper proposes a weighted voting strategy to assign an optimal weight to each base classifier, to better highlight the difference of each base classifier and further improve the detection performance. This section compares the FB2 detection scheme using the DE algorithm to optimize the weight with the FB2 without the right weight (that is, the weight of each base classifier is set to 1). The results are shown in Table 2.

As can be seen from Table 2, after assigning an optimal weight to each base classifier, the detection effect of each classifier is improved. In the absence of weight allocation, the FB2 ensemble detection scheme has achieved good results. However, after the differential evolution algorithm is used to assign the optimal weight to each base classifier, each detection index has been further improved. For example, the improvement degree of accuracy is at least 0.005 and at most 0.011. The *F*-score improved by a minimum of 0.004 and a maximum of 0.014. Therefore, it is necessary to assign an appropriate weight to each classifier.

5.4. Comparison with Base Classifier. This section evaluates the performance improvement of the FB2 for a single classifier. Therefore, FB2 was compared with the base classifier such as SVM, KNN, DT, LR, LDA, and MLP. For the parameters of each classifier, the default values were used. Table 3 shows the experimental results for each single base classifier and the FB2 detection results.

From Table 3, we draw the following conclusions. First of all, good performance has been achieved when only SVM, KNN, LR, and other single classification models are used as classifiers. Then, when using the FB2 scheme, performance improved significantly over the single classification model. It can be seen that, for all base classifiers, the FB2 scheme led to an improvement in all four performance metrics, with the improvement in accuracy ranging from 1.1% to 2.3%. The improvement in accuracy is 1.1% to 2.3%, and the improvement in *F*-score is 0.9% to 2.0%.

5.5. Comparison with Ensemble Classifier. This section compares the FB2 scheme with existing ensemble classifiers. For this purpose, the group with the worst results is selected from the results shown above and compared with the existing ensemble classifiers, such as RF, GBDT, and AdaBoost. For the parameters of each classifier, the default values were used. Table 4 shows the results. Although good performance has been achieved when only a base classifier is used as the classifier, compared with RF and other integrated classifiers, its performance is still inferior. However, after the FB2 integration scheme is used, its performance is further improved, which has a certain improvement compared with ensemble classifiers.

FB2 was compared with a variety of ensemble classifiers using the least effective base classifier DT. As can be seen from the four types of evaluation indicators shown in Table 4, although FB2-DT did not lead in all indicators, it was still the best in most cases. For example, accuracy and *F*-score both ranked first, with 0.978 and 0.977, respectively. Therefore, in general, FB2 is superior to the above three ensemble classifiers to a certain extent, which proves the effectiveness of the proposed scheme.

5.6. Comparison with Traditional Bagging Algorithms. This section compares the proposed FB2 scheme with the traditional bagging algorithm under a variety of single classifier models: SVM, KNN, DT, LR, and LDA. For a fairer comparison, default parameters are used for all classifiers, and the detailed results are shown in Table 5. Although the traditional bagging algorithm improves the performance of the single classifier, the performance of the FB2 scheme proposed in this paper is further improved. Among them, accuracy improved by 0.2% to 1.2%. And, *F*-score improved by 0.2% to 1.4%.

According to the data in Tables 1 and 5, the performance of FB2 is slightly worse than that of the traditional bagging algorithm in the case that the weighted voting strategy is not used, that is, the optimal weight is not given to each base classifier. However, after the optimal weight is allocated, FB2 has a significant improvement, surpassing the traditional bagging algorithm. This further proves that the scheme proposed in this paper is effective.

5.7. Model Sustainability Analysis. It is true that machine learning-based detectors are bound to suffer from deterioration, and many scholars are also studying related issues. In [46], to alleviate the classifier aging problem, they trained the classifier through API semantics which was extracted from the relationship graph of Android APIs. The extracted API semantics in the format of API clusters can be further used in Android malware classifiers to slow down aging. However, this paper uses five features for malware detection and does not use one feature of API alone, nor does each feature have a clear invocation relationship such as APIs, which prevents semantic extraction for each type of feature. And, the study by Xu et al. [47] is also based on the API detection scheme, which solves the aging problem of the classifier by constantly updating and evolving the API feature set. Both of these

Table 1: Performance of the new sampling strategy.

Base classifier	FB2 without weighted voting				Traditional bagging scheme			
	Accuracy	Precision	Recall	*F*-score	Accuracy	Precision	Recall	*F*-score
SVM	0.971	0.970	0.969	0.970	0.976	0.973	0.973	0.975
KNN	0.970	0.969	0.968	0.970	0.975	0.976	0.972	0.972
DT	0.969	0.967	0.969	0.969	0.970	0.969	0.971	0.973
LR	0.968	0.966	0.967	0.969	0.973	0.975	0.976	0.971
LDA	0.964	0.967	0.966	0.968	0.968	0.970	0.971	0.969
MLP	0.970	0.969	0.968	0.970	0.976	0.973	0.972	0.975

Table 2: Performance of the optimal weight.

Base classifier	FB2 without optimal weight				FB2 with optimal weight			
	Accuracy	Precision	Recall	*F*-score	Accuracy	Precision	Recall	*F*-score
SVM	0.976	0.973	0.973	0.975	0.984	0.983	0.986	0.985
KNN	0.975	0.976	0.972	0.972	0.980	0.980	0.982	0.981
DT	0.970	0.969	0.971	0.973	0.978	0.973	0.983	0.977
LR	0.973	0.975	0.976	0.971	0.983	0.981	0.984	0.985
LDA	0.968	0.970	0.971	0.969	0.979	0.981	0.978	0.979
MLP	0.976	0.973	0.972	0.975	0.982	0.981	0.985	0.983

Table 3: Comparison with multiple base classifiers.

Base classifier	Performance of the base classifier				Performance of the FB2			
	Accuracy	Precision	Recall	*F*-score	Accuracy	Precision	Recall	*F*-score
SVM	0.964	0.966	0.968	0.967	0.984	0.983	0.986	0.985
KNN	0.966	0.965	0.962	0.965	0.980	0.980	0.982	0.981
DT	0.967	0.963	0.969	0.966	0.978	0.973	0.983	0.977
LR	0.962	0.963	0.965	0.964	0.983	0.981	0.984	0.985
LDA	0.961	0.963	0.964	0.966	0.979	0.981	0.978	0.979
MLP	0.967	0.970	0.960	0.968	0.982	0.981	0.985	0.983

Table 4: Comparison with the ensemble classifier.

Ensemble classifier	Performance of the ensemble classifier			
	Accuracy	Precision	Recall	*F*-score
RF	0.976	0.981	0.974	0.977
GBDT	0.974	0.967	0.973	0.976
AdaBoost	0.977	0.970	0.976	0.975
FB2-DT	0.978	0.973	0.983	0.977

solutions use a single feature for detection, and the API is updated with the iteration of the Android version, while the other features basically do not change, so detection with just a single feature will face a more serious classifier failure problem. And, using many different types of features can effectively alleviate the degradation of the classifier. Five different features are used in this paper and are described in detail in Section 3.

And, the dataset used in this paper is AMD, and the samples are collected over a relatively close time span. Therefore, the dataset is divided into seven groups for testing according to the year, and each dataset is named BM plus the year. To ensure the fairness of the experiment, the worst-performing base classifier was selected. For a more comprehensive assessment, this paper uses the *F*-score metric i.e., both reusability and stability are based on the *F*-core. The detailed results are shown in Tables 6 and 7.

From Tables 6 and 7, it can be concluded that FB2 with multiple features achieves good classification results when training with old datasets and testing with new datasets. The average reusability of FB2 is about 96.7%, and the average stability is about 95.2%, both of which exceed DroidSpan. This shows that although DroidSpan based on a single dynamic feature is sustainable, FB2 based on multiple features is well adapted to malware updates and has good stability. In summary, the scheme proposed in this paper is sustainable under the condition of using many different types of features.

Table 5: Comparison with traditional bagging algorithms.

Base classifier	Performance of the traditional bagging				Performance of FB2 with optimal weight			
	Accuracy	Precision	Recall	*F*-score	Accuracy	Precision	Recall	*F*-score
SVM	0.971	0.970	0.969	0.970	0.984	0.983	0.986	0.985
KNN	0.970	0.969	0.968	0.970	0.980	0.980	0.982	0.981
DT	0.969	0.967	0.969	0.969	0.978	0.973	0.983	0.977
LR	0.968	0.966	0.967	0.969	0.983	0.981	0.984	0.985
LDA	0.964	0.967	0.966	0.968	0.979	0.981	0.978	0.979
MLP	0.970	0.969	0.968	0.970	0.982	0.981	0.985	0.983

Table 6: Reusability of FB2.

Base classifier	BM10	BM11	BM12	BM13	BM14	BM15	BM16
DT	0.964	0.965	0.966	0.969	0.963	0.967	0.971

Table 7: Stability of FB2 which trained on BM10 and tested on other datasets.

Base classifier	BM11	BM12	BM13	BM14	BM15	BM16
DT	0.956	0.943	0.942	0.952	0.954	0.965

6. Conclusion

A novel malware family-based bagging ensemble method (FB2) and Android malware detection scheme (FB2Dorid) were proposed in this paper. This detection scheme consists of the following steps. Firstly, APK files were decompiled to extract features. Then, the relief feature selection algorithm was applied to select a certain number of the most important features. Next, two sampling strategies, equal quantity sampling and based on family information sampling, are designed for improving the sampling strategy of the bagging integration algorithm. Finally, the base classifiers were combined with FB2.

In this paper, a variety of experiments are designed to verify the feasibility of the FB2 scheme. The experimental results indicate that the proposed FB2 scheme achieved good performance. In the future, the authors will consider designing new integration algorithms rather than just improving existing ones. Also, it is hoped that several sampling strategies proposed in this paper can really help scholars in this field. Let them use these sampling strategies in their own schemes to effectively mitigate the impact of sample imbalance and further improve the accuracy of malware detection and, secondly, to appeal to scholars engaged in Android malware research to pay more attention to the direction of data or sample imbalance. Although the growth of malware is rapid, it is still too small compared with benign applications, and the number of malicious samples in real datasets and reality detection will be very small, and the distribution gap between malware families will be large, so it is necessary to alleviate the sample imbalance. It is hoped that more scholars can be inspired by this paper, so as to put forward better schemes to eliminate the negative effects brought by the imbalance of samples.

In the future, authors consider selecting appropriate family information descriptors for each training subset and assigning different weights to the base classifiers trained under each sampling strategy.

Data Availability

The dataset used to support the findings of this study are free and publicly available on the Internet. The classification training data used to support the findings of this study are available from the website figshare. Readers can get the dataset through the following link https://doi.org/10.6084/m9.figshare.14329088.v1.

Conflicts of Interest

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

Acknowledgments

This research was supported by National Natural Science Foundation of China and Macau Science and Technology Development Joint Project (0066/2019/AFJ): Research on Knowledge-Oriented Probabilistic Graphical Model Theory Based on Multisource Data and project MF2009: Trusted Computing on Cross-Area Data.

References

[1] "Mobile operating system market share world-wide: global status," 2019, http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201904-%0A201904-bar.

[2] McAfee, "McAfee mobile threat report Q1, 2020 McAfee mobile threat report mobile malware is playing hide and steal," 2020.

[3] J. Wei, "New imbalanced bearing fault diagnosis method based on Sample-characteristic Oversampling TechniquE (SCOTE) and multi-class LS-SVM," *Applied Soft Computing*, vol. 101, 2020.

[4] A. Alzghoul, A. Jarndal, I. Alsyouf, and A. A. Bingamil, "On the usefulness of pre-processing methods in rotating machines faults classification using artificial neural network," *Journal of Applied and Computational Mechanics*, vol. 7, pp. 1–8, 2021.

[5] S. Sebbeh-Newton, P. E. A. Ayawah, J. W. A. Azure et al., "Towards TBM automation: on-the-fly characterization and classification of ground conditions ahead of a TBM using data-driven approach," *Applied Sciences*, vol. 11, no. 3, p. 1060, 2021.

[6] L. Liu, P. Wang, J. Lin, and L. Liu, "Intrusion detection of imbalanced network traffic based on machine learning and deep learning," *IEEE Access*, vol. 9, pp. 7550–7563, 2021.

[7] Z. Li, X. Zhang, J. Guo, and Y. Shang, "Class imbalance data-generation for software defect prediction," in *Proceedings of the Asia-Pacific Software Engeering Conference APSEC*, pp. 276–283, Putrajaya, Malaysia, December 2019.

[8] X. Zhang and D. Zhang, "Software defect prediction based on imbalanced datasets," *Journal of Applied Computing Research*, vol. 34, pp. 2027–2031, 2017.

[9] P. Faruki, V. Laxmi, A. Bharmal, M. S. Gaur, and V. Ganmoor, "AndroSimilar: robust signature for detecting variants of android malware," *Journal of Information Security and Applications*, vol. 22, pp. 66–80, 2015.

[10] M. Zheng, M. Sun, and J. C. S. Lui, "DroidAnalytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 163–171, Melbourne, Victoria, Australia, July 2013.

[11] J. Jung, H. Kim, D. Shin et al., "Android malware detection based on useful API calls and machine learning," in *Proceedings of the 2018 1st IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 175–178, Laguna Hills, CA, USA, September 2018.

[12] J. Jung, K. Lim, B. Kim, S. J. Cho, S. Han, and K. Suh, "Detecting malicious android apps using the popularity and relations of APIs," in *Proceedings of the IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering (AIKE 2019)*, pp. 309–312, Sardinia, Italy, June 2019.

[13] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, "Quick and accurate android malware detection based on sensitive APIs," in *Proceedings of the 2018 IEEE International Conference on Internet of Things*, pp. 143–148, Halifax, Canada, July 2018.

[14] 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 Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[15] C. Guo, J. Xu, L. Liu, and S. Xu, "MalDetector-using permission combinations to evaluate malicious features of android app," in *Proceedings of the IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 157–160, Beijing, China, November 2015.

[16] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Computing and Applications*, vol. 9, 2020.

[17] A. Pektaş and T. Acarman, "Deep learning for effective android malware detection using API call graph embeddings," *Soft Computing*, vol. 24, no. 2, pp. 1027–1043, 2020.

[18] X. Ge, Y. Pan, Y. Fan, and C. Fang, "AMDroid: android malware detection using function call graphs," in *Proceedings of the IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 71–77, Sofia, Bulgaria, July 2019.

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

[20] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, no. 8, pp. 1968–1982, 2020.

[21] W. Niu, R. Cao, X. Zhang, K. Ding, K. Zhang, and T. Li, "OpCode-level function call graph based android malware classification using deep learning," *Sensors*, vol. 20, no. 13, p. 3645, 2020.

[22] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A two-layer deep learning method for android malware detection using network traffic," *IEEE Access*, vol. 8, pp. 125786–125796, 2020.

[23] G. He, B. Xu, L. Zhang, and H. Zhu, "On-device detection of repackaged android malware via traffic clustering," *Security and Communication Networks*, vol. 2020, Article ID 8630748, 19 pages, 2020.

[24] K. Deepa, G. Radhamani, P. Vinod, M. Shojafar, N. Kumar, and M. Conti, "Identification of android malware using refined system calls," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 20, pp. 1–24, 2019.

[25] P. Vinod, A. Zemmari, and M. Conti, "A machine learning based approach to detect malicious android apps using discriminant system calls," *Future Generation Computer Systems*, vol. 94, pp. 333–350, 2019.

[26] A. S. M. Ahsan-Ul-Haque, M. S. Hossain, and M. Atiquzzaman, "Sequencing system calls for effective malware detection in android," in *Proceedings of the 2018 IEEE Glob. Commun. Conf. GLOBECOM 2018*, pp. 1–7, Abu Dhabi, UAE, December 2018.

[27] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "AndroDFA: android malware classification based on resource consumption," *Information*, vol. 11, no. 6, p. 326, 2020.

[28] K. Riad and L. Ke, "RoughDroid: operative scheme for functional android malware detection," *Security and Communication Networks*, vol. 2018, Article ID 8087303, 10 pages, 2018.

[29] Q. Fang, X. Yang, and C. Ji, "A hybrid detection method for android malware," in *Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 2127–2132, Chengdu, China, March 2019.

[30] R. B. Hadiprakoso, I. K. S. Buana, and Y. R. Pramadi, "Android malware detection using hybrid-based analysis & deep neural network," 2021.

[31] O. Faruk Turan Cavli and S. Sen, "Familial classification of android malware using hybrid analysis," in *Proceedings of the 2020 International Conference on Information Security and Cryptology (ISCTURKEY)*, pp. 62–67, Seoul, South Korea, December 2020.

[32] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian, "EC2: ensemble clustering and classification for predicting android malware families," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 2, pp. 262–277, 2020.

[33] M. Fan, J. Liu, X. Luo, K. Chen et al., "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 1890–1905, 2018.

[34] S. Niu, R. Huang, W. Chen, and Y. Xue, "An improved permission management scheme of android application based on machine learning," *Security and Communication Networks*, vol. 2018, Article ID 2329891, 12 pages, 2018.

[35] M. Yang, X. Chen, Y. Luo, and H. Zhang, "An android malware detection model based on DT-SVM," *Security and Communication Networks*, vol. 2020, Article ID 8841233, 11 pages, 2020.

[36] H.-J. Zhu, T.-H. Jiang, B. Ma, Z.-H. You, W.-L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for android," *Neural Computing and Applications*, vol. 30, no. 11, pp. 3353–3361, 2018.

[37] A. Egitmen, I. Bulut, R. C. Aygun, A. B. Gunduz, O. Seyrekbasan, and A. G. Yavuz, "Combat mobile evasive malware via skip-gram-based malware detection," *Security and Communication Networks*, vol. 2020, Article ID 6726147, 10 pages, 2020.

[38] X. Zhang, Y. Zeng, X. B. Jin, Z. W. Yan, and G. G. Geng, "Boosting the phishing detection performance by semantic analysis," in *Proceedings of the 2017 IEEE International Conference on Big Data (IEEE BigData 2017)*, pp. 1063–1070, Boston, MA, USA, December 2017.

[39] S. Y. Yerima and S. Sezer, "DroidFusion: a novel multilevel classifier fusion approach for android malware detection," *IEEE Transactions on Cybernetics*, vol. 49, no. 2, pp. 453–466, 2019.

[40] Androguard, A Full Python Tool to Play with Android Files: Anthony Desnos, https://github.com/androguard/androguard.

[41] Z. Zhu, Y.-S. Ong, and M. Dash, "Wrapper-filter feature selection algorithm using a memetic framework," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 1, pp. 70–76, 2007.

[42] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[43] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current android malware," in *Proceedings of the International Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, pp. 252–276, Bonn, Germany, July 2017.

[44] Google Play Store (Google), https://play.google.com/store/apps?hl=en.

[45] APKPure.com (APKPure Team), https://apkpure.com/cn/.

[46] X. Zhang, Y. Zhang, M. Zhong et al., "Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 757–770, Virtual Event, USA, November 2020.

[47] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: self-evolving android malware detection system," in *Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 47–62, Stockholm, Sweden, June 2019.