Hindawi
Security and Communication Networks
Volume 2021, Article ID 6689486, 12 pages
https://doi.org/10.1155/2021/6689486

Research Article

Permission Sensitivity-Based Malicious Application Detection for Android

Yubo Song[1,2] **Yijin Geng**[1,2] **Junbo Wang**[3,4] **Shang Gao**[5] **and Wei Shi**[1,2]

[1]*Key Laboratory of Computer Networking Technology of Jiangsu Province, School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China*
[2]*Purple Mountain Laboratories, Nanjing 211189, China*
[3]*School of Information Science and Engineering, Southeast University, Nanjing 211189, China*
[4]*National Mobile Communications Research Laboratory, Nanjing 211189, China*
[5]*Computing Department, The Hong Kong Polytheistic University, Hung Hom, Hong Kong*

Correspondence should be addressed to Yubo Song; songyubo@seu.edu.cn

Received 29 December 2020; Accepted 20 July 2021; Published 29 July 2021

Academic Editor: Liguo Zhang

Since a growing number of malicious applications attempt to steal users' private data by illegally invoking permissions, application stores have carried out many malware detection methods based on application permissions. However, most of them ignore specific permission combinations and application categories that affect the detection accuracy. The features they extracted are neither representative enough to distinguish benign and malicious applications. For these problems, an Android malware detection method based on permission sensitivity is proposed. First, for each kind of application categories, the permission features and permission combination features are extracted. The sensitive permission feature set corresponding to each category label is then obtained by the feature selection method based on permission sensitivity. In the following step, the permission call situation of the application to be detected is compared with the sensitive permission feature set, and the weight allocation method is used to quantify this information into numerical features. In the proposed method of malicious application detection, three machine-learning algorithms are selected to construct the classifier model and optimize the parameters. Compared with traditional methods, the proposed method consumed 60.94% less time while still achieving high accuracy of up to 92.17%.

1. Introduction

Nowadays, Android operating system occupies 85% of the market share and has become the most popular mobile operating system. However, owing to the increase in usage and the source's openness, the Android system becomes the target of malware. According to the report on China's mobile phone security status in the first half of 2020, 360 Security Brain intercepted approximately 1.048 million new malware samples on the mobile terminal, approximately 6 thousand new ones added every day. These malicious applications hidden in some application stores have access to users' privacy, resulting in information leakage of private data and economic loss, affecting the Android ecosystem's healthy development [1, 2]. Therefore, the research on efficient malware detection methods for application stores is very crucial. However, the traditional detection methods based on permission features mostly have the following problems:

(i) The legality of requesting the same permission is different in various categories of applications. For example, requesting permission to read contacts is a legal behavior in social chat applications, but it is usually regarded as a malicious behavior in photo-taking applications. In general, putting all types of Android applications in the same data set together may cause misjudgment.

(ii) Many malicious behaviors need to call multiple permission combinations. For example, the

malicious behavior of uploading contact information to a website requires calling READ_CONTACTS and Internet permissions. Therefore, analyzing the information of permission invocation, ignoring the influence of some specific permission combination on the malware detection will result in the decrease of the detection accuracy.

(iii) Some permission features cannot effectively distinguish between benign applications and malicious applications. Taking such permission features into consideration will lead to many invalid features, which will reduce the accuracy of detection and cost more detection time.

To solve the above problems, considering the coarse-grained authorization mechanism of the Android system, we present an Android malware detection method based on permission sensitivity. This method screens the permission features and permission combination features based on permission sensitivity to obtain the sensitive permission feature sets of various applications. The weight allocation method is then used to quantify the sensitive permission features, thus realizing malware detection. The main contributions of our work are summarized as follows:

(i) An automatic classification method based on semantic analysis is proposed. The application's primary function is identified through the semantic analysis of the application description text, and then the label category can be attached to the application.

(ii) Considering the sensitivity of permission features and permission combination features under various category labels are different, the permission feature set is filtered to obtain the application's sensitive permission feature set to perform malicious application detection.

(iii) We propose a method of processing permission features based on weight allocation. Considering that permission features have different abilities to distinguish malicious and benign applications, the corresponding weights are assigned to different features. Thus application permission request information can be converted into digital characteristics for malicious application detection.

(iv) The experimental results show that the accuracy rate of the semantic analysis-based classification method is 91.65%. The accuracy rate of the malware detection method is 92.17%, and this method only requires 0.1256 seconds to detect a single sample on average, which is 60.94% less than the traditional detection method.

2. Related Works

The classification methods of Android applications are mainly divided into two types: manual classification and automatic classification. The manual classification method determines the application's basic functions by the application store staff through the application's trial run; thereby, the application's label can be determined. This method's operation complexity is low, but the time cost and labor costs are enormous, and the error rate is inevitable.

Automatic classification is the process that the classifier model reads and analyzes the code file or other useful information of the application to determine the application's category according to the predefined category label set. Compared with the manual classification method, the automatic classification method, with the help of powerful computing and storage capacity of the computer, eliminates manual trial operation and simulated interactive operation, which dramatically improves application category determination efficiency. The application category automatic determination method is mainly aimed to find the key features in the static code files of Android applications for processing and analysis, which is also called the static analysis method. Burguera et al. [3] parsed the Android application software package to extract static features, combined them with the APK folder's size, thus can perform feature filtering and fusion to form joint features for classification. After that, the joint features are formed for classification. Yuan et al. [4] parsed and decompiled the APK file, extracted the API call information and the string as features for classification. However, decompiling the APK file of the software package to extract features for classification requires a long work cycle, which affects the classification efficiency. Kutlay and Karaduzovic-Hadziabdic [5] and Ahmad et al. [6] both used static analysis to extract features and construct classifiers for classification through machine-learning algorithms. However, these methods usually use an unsupervised learning clustering algorithm to construct classifiers, dividing the static features into several clusters, not determining the specific category label in the Android App store.

Several methods have been proposed for Android malware detection, including dynamic analysis, static analysis, and hybrid analysis. Each method has its merits and shortcomings.

Dynamic analysis refers to the methods that deploy the application on the emulator or actual device to execute it entirely. Simulating the interaction between users and applications can determine whether there is any malicious behavior [7, 8]. Wu et al. [7] used the system call frequency and system call dependence to construct joint features for malicious detection, and the accuracy rate was 93%. Jeon et al. [8] performed dynamic analysis on malware to extract related malicious behaviors such as system calls. They used them as a basis for discrimination to compare the application's behavior data to be tested, determining the application's malicious degree. Although these malware detection methods based on dynamic analysis have high accuracy, they cannot be widely used because of the difficulty of execution and high cost of time.

The static analysis method realizes detection by analyzing the application's static code and does not need to run the application entirely. Because of omitting the running process, the static analysis method consumes fewer hardware resources and time, so it is a relatively efficient and fast detection method. Lindorfer et al. [9] combined the

permission request information of the application with the sensitive API call information to form a joint feature for malicious detection. Guen et al. [10] extracted seven functions from the APK file by analyzing the list file and DEX file. These functions enrich the extracted information to express the application's characteristics to investigate whether the application under test has malicious behavior. However, these methods ignore the influence of application category on the accuracy, that is, the legality of the same permission request information is different in different types of applications. Besides, static analysis methods are hard to defend against obfuscation and repackaging.

Hybrid analysis refers to the combination of static analysis and dynamic analysis for malicious application detection. Therefore, hybrid analysis has the advantages of both. This method analyzes the APK folder of the Android application package and various application runtime behaviors, which is considered a comprehensive analysis method. However, the hybrid analysis also inherits the apparent disadvantages of dynamic analysis, which will lead to excessive consumption of Android system resources and a large amount of time waste [11].

In summary, dynamic analysis, static analysis, and hybrid analysis all have certain drawbacks, and they are not suitable for large-scale malware detection in the Android application market.

3. Detection Methodology

The process of android malware detection based on permission sensitivity is illustrated in Figure 1, and the details are provided as follows:

(i) Data set preprocessing: The data set preprocessing includes two parts. One is to obtain the application category label; the other is to obtain the application permission invocation information. The category label of the application is determined by the semantic analysis based on the application description text. The permission call information is gathered by decompiling the APK file of the Android application installation package and parsing the relevant files.

(ii) Permission feature processing: Permission feature processing consists of two parts, the extraction of sensitive permission feature set and the feature quantification based on the weight allocation method, which is convenient for inputting the classifier model training. The acquisition of the sensitive permission feature set is obtained by filtering permission features and permission combination features based on permission sensitivity. The feature quantification is based on the weight allocation method, which quantifies the application sensitive permission request information into the digital features to measure the application's malicious degree.

(iii) Classifier model construction: The classifier model is constructed based on the appropriate machine-learning algorithm. The malicious detection of the application under test is completed by training sensitive digital features.

3.1. Permission Sensitivity. The permission features and permission combination features are filtered based on the permission sensitivity in detecting malicious applications. A representative set of sensitive permission features is obtained for each type of application. Besides, to accurately represent the distinguishing strength of different permissions on malicious applications, the feature set is divided into two categories corresponding to the permission sensitivity, namely, lost permission and overload permission. In this way, different weights are assigned to the two types of permissions based on the weight allocation method, realizing the quantification of sensitive permissions. For the permission set $P_{\mathrm{er}} = P_{\mathrm{er1}}, P_{\mathrm{er2}}, \ldots, P_{\mathrm{er}k}$, there are the following definitions:

3.1.1. Permission Matrix. For the permission call information of a single application sample under the category label, the permission vector is obtained by comparing it with the permission set:

$$\overrightarrow{\mathrm{PV}_{t_i j}} = \left(\mathrm{pv}_1, \mathrm{pv}_2, \ldots, \mathrm{pv}_k\right)^T. \tag{1}$$

Each position in the vector can only take 0 or 1. The value of 0 means that the application does not call the permission of the corresponding position in the permission set. Otherwise, the value is 1. The permission matrix is obtained by multiplying the permission vector with its transposition matrix:

$$\mathrm{PM}_{t_i j} = \overrightarrow{\mathrm{PV}_{t_i j}} \times \overrightarrow{\mathrm{PV}_{t_i j}}^T = \left[\mathrm{pm}_{mn}\right] 0 \le m \le k, \quad 0 \le n \le k. \tag{2}$$

The value of each position in the permission matrix is 0 or 1. pm_{ii} $(0 \le i \le k)$ on the diagonal line of the matrix represents the request status of the permission feature with the sample application. The value of 1 means that the application has called the ith position permission in the set P_{er}, otherwise, the value is 0. pm_{ij} $(0 \le i \le k, 0 \le j \le k, i \ne j)$ on the nondiagonal position of the matrix indicates the invocation of the combination of the ith permission and the jth permission feature in the sample application. A value of 0 means that the application does not call the permission combination, and a value of 1 means that the sample application calls the permission combination.

3.1.2. Average Permission Matrix. The average permission matrix of benign applications under t_i category label is obtained by summing the permission matrix of all benign applications under t_i category label and dividing by the total number of benign applications:

$$\overline{\mathrm{PM}_{t_i B}} = \frac{\sum_{j=1}^{\mathrm{size}\left(t_i B\right)} \mathrm{PM}_{t_i B_j}}{\mathrm{size}\left(t_i B\right)}. \tag{3}$$

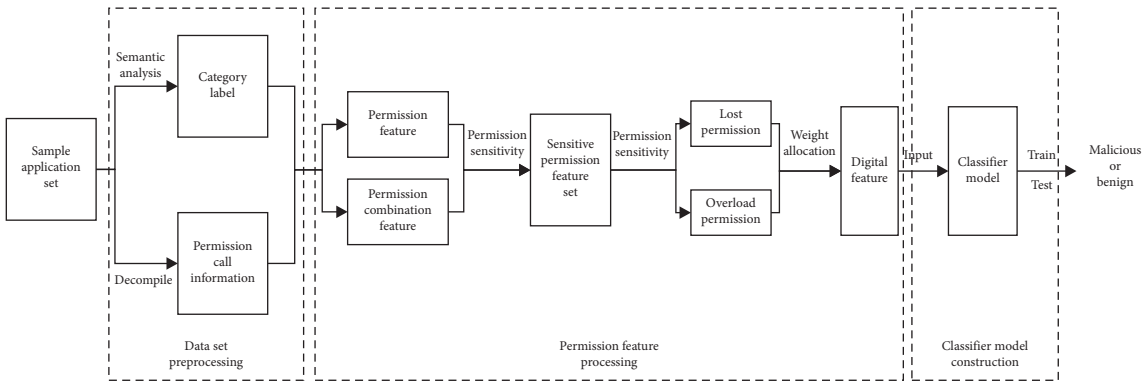

Figure 1: Malicious application detection process.

The average permission matrix of malicious samples is obtained by the same operation:

$$\overline{\mathrm{PM}_{t_iM}} = \frac{\sum_{j=1}^{\mathrm{size}\left(t_iM\right)} \mathrm{PM}_{ti}M_j}{\mathrm{size}\left(t_iM\right)}, \tag{4}$$

size (x) is a function used to calculate the number of elements in the set x. The number in each position of the average permission matrix represents the call frequency of permission features or combination features.

3.1.3. Permission Sensitivity. The average permission matrix of benign samples based on t_i category label is subtracted from that of malicious samples to get the permission sensitivity matrix under the category label. The value of each position in the matrix is the permission sensitivity (PS). Permission sensitivity is a physical quantity to measure the degree of distinguishing between malicious and benign applications. Under the category label t_i, the permission sensitivity of the permission feature m, and the permission sensitivity of the combination feature of permission m and permission n, are respectively expressed as follows:

$$\mathrm{PS}_{t_imm} = \overline{\mathrm{PM}_{t_iB_{mm}}} - \overline{\mathrm{PM}_{t_iM_{mm}}}, \quad 0 \le m \le k, \tag{5}$$

$$\mathrm{PS}_{t_imn} = \overline{\mathrm{PM}_{t_iB_{mn}}} - \overline{\mathrm{PM}_{t_iM_{mn}}}, \quad 0 \le m \le k, 0 \le n \le k, m \le n. \tag{6}$$

The greater the absolute value of the feature's permission sensitivity, the greater the discrimination between benign and malicious applications. For each type of application, permission features and combination features can be filtered to eliminate the redundant features to obtain the sensitive permission feature set based on the permission sensitivity. At the same time, to accurately quantify the application of the sensitive permission feature set, the sensitive permission feature set is further subdivided based on the permission sensitivity, and different types of features are given different weights, which can effectively represent the discriminate strength of features against malicious samples.

3.2. Automatic Classification Based on Semantic Analysis. The same permission request has different legality judgments in various applications, so it is necessary to classify applications to detect malicious applications.

Semantic analysis is used to extract the category label based on application description text. Before submitting an application package to the app store for release, the developer must fill in the application function description text, publisher, and other relevant information. The filling of this information is directly related to the publisher's digital signature. It will be reviewed by the staff of the app store, which has high credibility and authenticity. The specific steps are shown in Figure 2.

The Android application description text obtained by web crawler technology usually contains null characters, special meaning symbols, and so on, which disturb the semantic analysis. It is necessary to use Unicode encoding and other methods to preprocess the application description text to reduce noise. Simultaneously, to classify applications based on description text, it is necessary to predefine the set of category labels.

The preprocessed text set needs to be extracted with feature words. The Term Frequency-Inverse Document Frequency algorithm (TF-IDF) does not involve iterative calculation and word comparison, which saves hardware memory space and extraction time. The effect of feature word extraction is very prominent. Therefore, the TF-IDF algorithm is used to extract feature words from the description text of each type of application [12]. Algorithm 1 gives its pseudocode.

In order to process the relationship between a single feature word and a set of feature words into useful features that can be recognized and read by a machine-learning algorithm, it is necessary to quantize the feature words so that the text information can be transformed into a meaningful number vector or array [13]. Bag of words (BOW) is the most common method of word vectorization. Its main idea is to take a high-order vector whose dimension is equal to the size of the feature word set, and each position of the vector represents the number of times the corresponding word appears in the word set [14]. The word bag model considers the influence of the occurrence times of

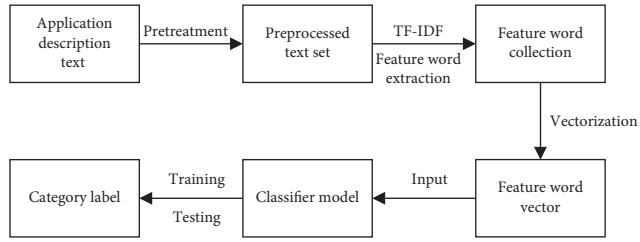

Figure 2: Process of application category determination based on semantic analysis.

Input: T is the label collection, D is the application description text collection
Output: Feature word collection Words
1: initialize an Array of Words
2: **for** each $t_i \in T$ **do**
3: initialize a map Map_{t_i}
4: initialize an array doc_{ij}
5: **for** each $\text{app}_{ij} \in t_i$ **do**
6: $\text{doc}_{ij} = \text{pretreatment}\ (D_{ij})$
7: $\text{words}_{ij} \longleftarrow \text{TF} - \text{IDF}\ (\text{doc}_{ij})$
8: **for** each $\text{word}_k \in \text{words}$ **do**
9: **if** $\text{word}_k \in \text{Map}_{t_i}$ **then**
10: $\text{value}_{\text{word}_k} \longleftarrow \text{value}_{\text{word}_k} + 1$
11: **else** Map_{t_i}.push (key: word_k, value: 1)
12: **end if**
13: **end for**
14: **end for**
15: Words.push (Map_{t_i})
16: **end for**
17: **return** Words

Algorithm 1: Feature word set extraction algorithm of description text based on TF-IDF.

feature words on feature vectors. However, due to the uncertainty of the number and frequency of feature words in a single text, the difference between feature vectors in various texts is too high. The complexity of the subsequent calculation is significantly increased. Therefore, the bag of words model is improved: use the normalized exponential function to process the digital vector based on the bag-of-words model so that the sum of all the digits of the vector is equal to 1. The specific process is as follows:

The extracted feature word set is Words, and the set size is n. Comparing the feature words extracted by a specific application description text d with the feature word set, the vector can be obtained using the bag of words model:

$$\overrightarrow{V_d} = (v_1, v_2, \ldots, v_n)^T. \tag{7}$$

Use the exponential function to normalize the vector:

$$V_{\text{si}} = \frac{e^{vi}}{\sum_{j=1}^{n} e^{vj}}. \tag{8}$$

The normalized vector is

$$\overrightarrow{V_{\text{sd}}} = (v_{s1}, v_{s2}, \ldots, v_{sn})^T. \tag{9}$$

In this way, the feature words can be converted into digital vectors, which can then be input into the classifier model for training and predicting category labels.

3.3. Extraction of Sensitive Permission Feature Set. In order to reduce the detection time and save memory space, the permission features and combination features are filtered based on the permission sensitivity to obtain the sensitive permission feature set based on the category label.

The Android operating system initially designed more than 400 permissions for developers to use. Some permissions can only be called by the system, and ordinary developers have no right to use them. This type of permission is the first to eliminate in the study. Simultaneously, in the latest version of the Android log, it was mentioned that some permissions will rarely be used by the regular application and will be discarded in future versions, such as the SET_PREFERRED_APPLICATIONS permission, so these permissions are filtered. Use $P_{\text{er}} = \{P_{\text{er1}}, P_{\text{er2}}, \ldots, P_{\text{erk}}\}$ to indicate the remaining set of permissions after filtering.

The benign application samples B_{t_i} and malicious application samples M_{t_i} under the label t_i are obtained from the public data set. Related operations such as decompiling the APK file are carried out to get the application's permission call information. For a single application sample app_{ij} under the t_i category label, the binary permission vector $\overrightarrow{\text{PV}_{t_{ij}}}$ is

obtained by comparing the permission call information with the permission set.

The transposition vector of the feature vector is denoted as $\mathrm{PV}_{t_{ij}}T$, according to formula (2), the feature matrix $\underrightarrow{\mathrm{PM}}_{t_i j}$ of $k \times k$ dimension is obtained by multiplying with $\overrightarrow{\mathrm{PV}}_{t_{ij}}$.

According to formulas (3) and (4), the average permission matrix $\underrightarrow{\mathrm{PM}}_{t_i B}$ of benign samples and the average permission matrix $\underrightarrow{\mathrm{PM}}_{t_i M}$ of malicious samples can be calculated. The diagonal number in the average permission matrix represents the permission coverage of the corresponding permission feature. The nondiagonal number represents the permission coverage of the corresponding permission combination feature. Permission coverage is a measure of the frequency of calling a specific permission feature or permission combination feature under a single category.

When the average permission matrix of benign application samples is subtracted from that of malicious application samples under the t_i category label, the value is the permission sensitivity of corresponding features. According to formula (5), PStimm, the sensitivity of permission feature m to malicious samples and benign samples can be calculated. According to formula (6), $\mathrm{PS}_{t_i mn}$, the sensitivity of permission combination features of permission m and permission n under the t_i category label can be calculated.

The permission sensitivity indicates the ability of the feature to distinguish between benign and malicious applications. The higher the absolute value of permission sensitivity, the more pronounced the discrimination between benign and malicious applications. Therefore, for all the permission features and permission combination features under the t_i category label, the permission sensitivity $x_1, x_2, \ldots, x_n$ is arranged from large to small according to the absolute value. Set threshold μ:

$$\mu = \min_x \frac{\sum_{j=1}^{n} \left(x - x_j\right)^2}{(n-1)^2}. \tag{10}$$

The features whose absolute value of permission sensitivity is greater than or equal to the threshold are put into the sensitive permission feature set, which means the permission feature can distinguish benign and malicious applications more effectively to increase the detection accuracy. If the absolute value of the permission sensitivity is less than the threshold value, the feature which wastes time and space resources will be eliminated. When faced with conflicts between permission features and permission combination features, the permission combination features are considered to be more effective in distinguishing malicious and benign, and the permission combination features are added to the sensitive permission feature set, while the permission features are discarded, and the sensitive permission feature set of the t_i category label is obtained. After repeated operations, the sensitive permission feature set corresponding to all category labels can be obtained.

3.4. Quantification of Sensitive Permission. The permission request information of the application to be tested is compared with the sensitive permission feature set based on the category label. The result is represented by digital features, which better reflects the malicious degree of the application. This process is the quantification of sensitive permission features.

In traditional malware detection methods, binary methods are usually used to process features. Use 0 to indicate that the permission is not invoked, and 1 indicates that the permission is invoked. Thus the high-dimensional vectors of 0 and 1 are formed and input into the classifier. But the binary method has apparent shortcomings, for the permission group set per, assuming that the sensitivity of permission i is greater than that of j, it means that the distinguishing power of permission feature i for malicious and benign samples is higher than that of permission feature j. If expressed by the binary method, their corresponding positions will be assigned as 1 or 0, which cannot reflect the degree of distinguishing malicious and benign applications. The neglect of the relationship will affect the detection results, so the weight allocation method is proposed to quantify the features.

First, two types of permissions are defined: lost permission and overload permission.

3.4.1. Lost Permission. When calculating the permission sensitivity, the values are positive and negative. When the permission sensitivity of features is positive, under the category label, the permission coverage of benign application samples is greater than that of malicious application samples. Such features are defined as lost permissions (LP). Under the category label, the probability that the permission feature j appears in the benign application sample is higher than that in the malicious application sample. The benign application usually calls the permission feature j, while the malicious application rarely calls the permission feature j, which is the lost permission.

3.4.2. Overload Permission. The permission features with negative permission sensitivity under the same category label. That means the probability of permission feature j appearing in the malicious application sample is higher than that in the benign application sample. The malicious application usually calls permission feature j, while benign application rarely calls permission feature j, which is overload permission.

Sensitive permissions are divided into two categories because they have different influences on malware detection. The lost permission means that the application under test does not call one or more key permission features than benign applications. For example, a photographing application does not apply for calling camera permission. Under normal circumstances, it can be considered that the application under test may not have the normal functions of other applications under the category label. Still, there will be no other malicious behavior. Overload permission means that the application under test calls one or more key permission features, such as a shopping application calling contact information, compared with the benign application. This

extra behavior should be considered as malicious behavior beyond the acceptable range. Therefore, the discrimination between malicious and benign of the overload permission should be higher than that of the lost permission. When α and β are used to measure the discrimination of overload permission and the lost permission for malicious samples, $\alpha \geq \beta$.

The main disadvantage of the binary method is that it cannot reflect the difference in permission sensitivity. Therefore, all permission features under the label are normalized according to the sensitivity. Each permission feature is assigned a weight to measure the ability to distinguish benign and malicious application samples. Assuming that the sensitive permission feature set based on the t_i category label is $(P_1, \mathrm{PS}_{t_i 1})$, $(P_2, \mathrm{PS}_{t_i 2}), \ldots$, $(P_n, \mathrm{PS}_{t_i n})$, where P is the specific permission, and PS represents the sensitivity of the permission. For permission P_k, the weight can be calculated as follows:

$$w_k = \frac{\left|\mathrm{PS}_{t_i k}\right|}{\sum_{m=1}^{n}\left|\mathrm{PS}_{t_i m}\right|}. \tag{11}$$

For all features in the sensitive permission feature set, the weight vector W is obtained:

$$\overrightarrow{W} = (w_1, w_2, \ldots, w_n)^T. \tag{12}$$

Combining overload permissions and lost permissions, the above formula is further processed:

$$\overrightarrow{\mathrm{PW}} = (\mathrm{pw}_1, \mathrm{pw}_2, \ldots, \mathrm{pw}_n)^T,$$
$$\mathrm{pw}_i = \begin{cases} w_i \times \alpha, & p_i \in \mathrm{OP}, \\ w_i \times \beta, & p_i \in \mathrm{LP}. \end{cases} \tag{13}$$

Assuming that the application permission call vector obtained by the binary method is $\overrightarrow{Bin_i}$, then the digital feature obtained by the weight distribution method is

$$W_i = \overrightarrow{Bin_i} \cdot \overrightarrow{\mathrm{PW}}. \tag{14}$$

The pseudocode of the specific weight allocation algorithm is shown in Algorithm 2.

3.5. Construction of Classifier Model for Malware Detection. A classifier is constructed to process the digitized sensitive features to complete the malware detection based on the machine-learning algorithm. Among the most commonly used traditional machine-learning algorithms, the random forest algorithm has excellent performance compared to other traditional machine-learning algorithms [15]. It takes less time to process large-scale samples. Therefore, the random forest algorithm is used to construct a classifier model for Android malware detection. The detailed process is shown in Algorithm 3.

4. Experimental Results and Analysis

In this experiment, 3400 Android malicious application samples from the VirusShare [16] and Arp et al. [17] data sets, 4500 normal applications as the benign Android samples were selected from the Huawei application market through crawler technology. 80% of the samples are used as the training set, and 20% of the samples are used to test the classifier model's performance. Before classifying the applications, the application category label set is predefined: navigation, online shopping, weather, finance, education, catering, camera, music, social, communication, news, video, reading, tourism, sports, office, tools, and game.

4.1. Evaluation Metrics. Suppose TP and TN are the numbers of benign applications and malicious applications correctly identified, FP and FN are the numbers of misjudged malicious applications and benign applications, respectively. Then the commonly used evaluation metrics of two classification problems are accuracy A, precision P_{re}, recall R_{ec} and F_1:

$$A = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}},$$
$$P_{\mathrm{re}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}},$$
$$R_{\mathrm{ec}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{15}$$
$$F_1 = \frac{2 \times \mathrm{TP}}{2 \times \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}.$$

For the multiclassification problem, the concept of macro average is introduced, and the evaluation metrics of each category label are calculated by arithmetic average: macro precision P_{macro}, macro recall R_{macro}, and macro $F_{1\mathrm{macro}}$ to measure the performance of the detection method.

$$P_{\mathrm{macro}} = \frac{1}{n}\sum_{i=1}^{n} P_i,$$
$$R_{\mathrm{macro}} = \frac{1}{n}\sum_{i=1}^{n} R_i, \tag{16}$$
$$F_{1\mathrm{macro}} = \frac{2 \times P_{\mathrm{macro}} \times R_{\mathrm{macro}}}{P_{\mathrm{macro}} + R_{\mathrm{macro}}}.$$

The classifier model includes two processes: training and testing. Symbol t_r is used to represent the time taken by the classifier to complete the sample training, the symbol t_p to represent the time taken for the classifier to complete the sample test, the symbol t_s to represent the total time for training and prediction, and the symbol $\bar{t}$ to represent the time consumed for processing a single sample.

4.2. Analysis of Results. In the experiment of category label determination, k-nearest neighbor (KNN) algorithm [18], Naive Bayes Model (NBM) [19], and TextCNN algorithm [20] are used to construct classifier models for comparison. In the malware detection experiment, the value of α/β in the model is tuned. The optimized detection model is compared with the traditional detection model to check accuracy and efficiency.

Input: Test is the application to be tested
Output: The weight of permission
for each app ∈ Test **do**
  initialize a value result = 0
  $\text{Bin}_{\text{app}} \longleftarrow$ Binary (app)
  for each $i \in \text{PS}_{\text{Groups}}$ **do**
    $W_i \longleftarrow$ Normalized (W_i)
    if $i \in$ LP **then**
      result $\longleftarrow$ result + $\text{Bin}_{\text{app}_i} * W_i * \alpha$
    else
      result $\longleftarrow$ result + $\text{Bin}_{\text{app}_i} * W_i * \beta$
    end if
  end for
  return result
end for

Algorithm 2: The indication of sensitive permission call information based on the weight allocation method.

Input: data is the sample data set, s is the data set extraction ratio, DT is the decision tree collection, x is the attribute classification method
Output: result Malware detection result
initialize a value num_b
initialize a value num_m
for each $dt \in$ DT **do**
  initialize a Boolean temp
  $\text{data}_{dt} \longleftarrow \text{data} * s$
  temp$\longleftarrow$DT(data_{dt}, x)
  if temp = true **then**
    $\text{num}_b \longleftarrow \text{num}_b + 1$
  end if
  if temp = false **then**
    $\text{num}_m \longleftarrow \text{num}_m + 1$
  end if
end for
if $\text{num}_b > \text{num}_m$ **then**
  result$\longleftarrow$Benign
end if
if $\text{num}_b < \text{num}_m$ **then**
  result$\longleftarrow$Malicious
end if
return result

Algorithm 3: Construct a classifier based on the random forest algorithm to perform malware detection.

4.2.1. k-Nearest Neighbor Classifier. The accuracy of the classifier model constructed by the k-nearest neighbor algorithm mainly depends on two aspects: the value of k and the choice of distance calculation methods. In this experiment, these two parameters were tuned separately, and the results are shown in Figure 3.

Figure 3(a) shows the evaluation metrics A, P_{macro}, R_{macro}, and $F_{1\text{macro}}$ of KNN classifier model using Euclidean distance under different k values. It can be seen that when $k < 15$, the performance of classifier accuracy is generally low and fluctuates significantly at that time. When $k = 15$, all the classifier's evaluation indexes reached the best, which were 0.8842, 0.9016, 0.8749, and 0.8853, respectively. When $k > 15$, the indicators of the model declined slowly. Therefore, when k is 15, the k-nearest neighbor classifier model has the best performance.

Figure 3(b) shows the k-nearest neighbor classifier model's performance results using Euclidean distance and Manhattan distance, respectively. From the graph, we can find that the A, P_{macro}, R_{macro}, and $F_{1\text{macro}}$ of the classifier using Manhattan distance formula are generally low, which is far less than the classifier's performance using Euclidean distance under the same conditions. According to the above experimental results, when the k-nearest neighbor classifier model has the best performance, the value of k should be 15, and the distance calculation method should be Euclidean distance.

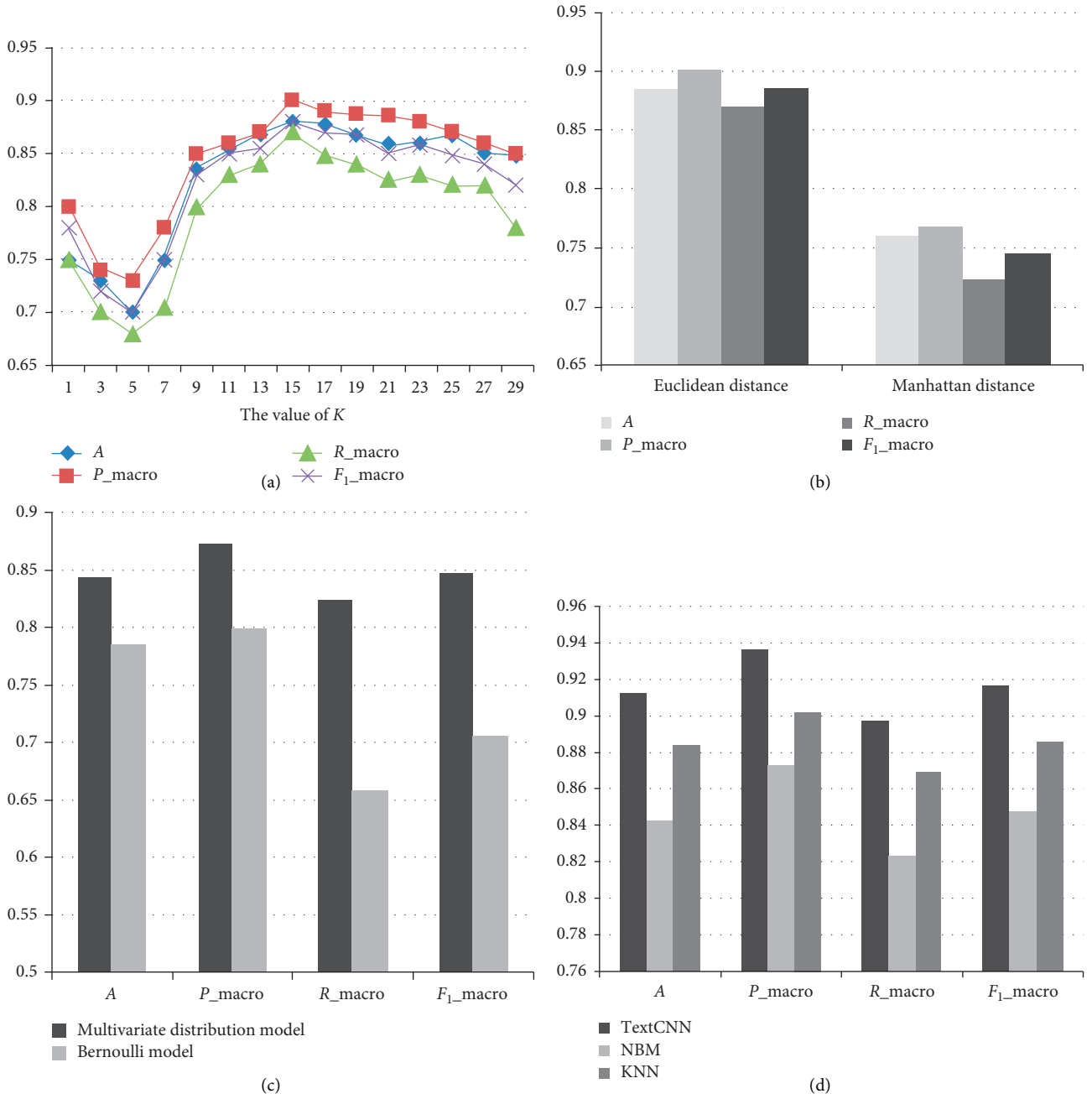

Figure 3: Experimental results of different classifiers. (a) KNN classifier with different k values, (b) KNN classifier with different distance formulas, (c) NBM classifier under different models, and (d) performance of different classifier models.

4.2.2. Naive Bayes Model Classifier. The classifier model's parameters based on the Naive Bayes Model are optimized to explore the distribution model (Bernoulli Model, Multivariate Distribution Model), which makes the accuracy performance of the classifier model the best. The results are shown in Figure 3(c). It can be seen from Figure 3(c) that under the same data set, the evaluation metrics obtained using the multivariate distribution model are better than those using the Bernoulli model, which is of great help to improve the accuracy performance of the classifier.

4.2.3. Performance Comparison of Three Classifier Models. TextCNN algorithm is a representative algorithm model to solve the problems of text classification and semantic analysis. Compared to the k-nearest neighbor algorithm classifier model, the Naive Bayes algorithm classifier model, and the TextCNN algorithm classifier model with the performance, the results are shown in Figure 3(d). The graph shows that the A, P_{macro}, R_{macro}, and $F_{1\text{macro}}$ of the classifier constructed based on the TextCNN algorithm are 0.9165, 0.9368, 0.8971, and 0.9127, respectively, which are better than the other two models. It can be seen that in the data sets

Table 1: Efficiency performance of three classifier models.

	t_r (s)	t_p (s)	t_s (s)	$\bar{t}$ (s)
KNN	2202.81	339.63	2542.44	0.2472
NBM	1226.97	308.74	1535.71	0.1677
TextCNN	2474.62	1213.95	3688.57	0.5090

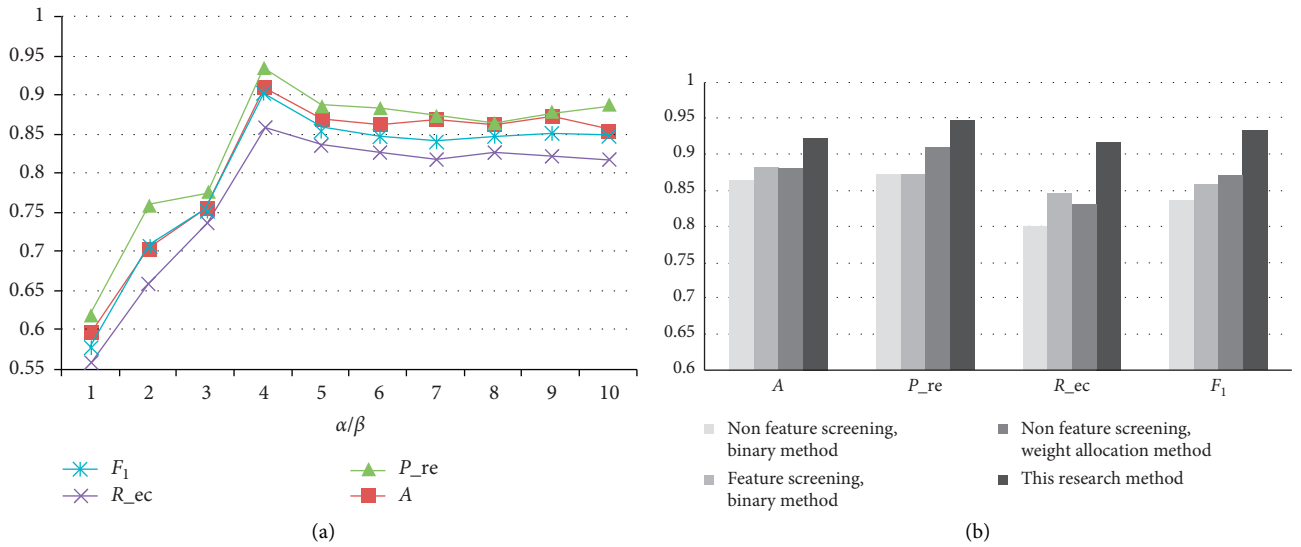

Figure 4: Experimental results of malware detection. (a) α/β optimization experiment results and (b) accuracy performance results of four methods.

and application scenarios of this study, the deep learning algorithm is more suitable than the traditional machine-learning algorithm in constructing a classifier model to complete the automatic application category determination based on semantic analysis.

Table 1 shows the performance of the three models on classification speed. It can be seen from Table 1 that in the total training and prediction time, the TextCNN classifier model with the best accuracy performance takes the most time, reaching 3688.57 seconds. In terms of the average training and prediction time of a single sample, the Naive Bayesian Model performs best, and it only takes 0.1677 seconds to process a sample. Considering that the ratio of training data set to test data set in this experiment is 4 : 1, from the approximate results, the ratio of training time to test time of classifier constructed based on Naive Bayesian algorithm is equal to the ratio of training data set to training data set, that is, the time complexity can be considered as $O(n)$. Although the total time consumed by the TextCNN model is relatively large, the training and prediction time is relatively large, and the time complexity is similar to $O(n)$. This means that when the data set is vast, the disadvantage of using the TextCNN algorithm to construct a classifier will be significantly reduced, which can even become a more efficient algorithm.

In summary, the accuracy of the three algorithms for application classification is above 85%. In terms of accuracy, the classification accuracy based on the TextCNN algorithm reached 91.65%, and the macro precision reached 93.68%, which is better than the other two classifier models. In terms of efficiency, the classifier model based on the Naive Bayes model takes only 0.1677 seconds to process a single sample, which is suitable for application stores with high-efficiency classification.

4.2.4. Determination of Parameters in the Malware Detection Model. In quantifying sensitive permission features based on weight allocation, the ratio measures the ability of permission features to distinguish malicious and benign samples. Figure 4(a) shows the results of the accuracy rate A, precision rate P_{re}, recall rate R_{ec}, and F_1 value of the malware detection method as the parameters $\alpha\beta$ change. It can be seen from the figure that when the parameter is small, especially when $\alpha\beta = 1$, the classifier's evaluation metrics are not ideal, and the detection accuracy of malicious applications is only 59.76%. As the parameter $\alpha\beta$ gradually increase, each evaluation metric curve rises with a larger slope, and the detection performance is significantly improved. When $\alpha\beta = 4$, each evaluation metric of the classifier reaches the maximum value. Therefore, the optimal value of the $\alpha\beta$ should be 4.

4.2.5. Performance Comparison of Four Different Malicious Detection Models. The Android malware detection method proposed in this paper is compared with the detection model without feature screening and using the binary method, the detection model with feature screening and using the binary method, and the detection model without feature screening and using the weight allocation method to compare the

Table 2: Efficiency performance of four detection models.

	t_r (s)	t_p (s)	t_s (s)	$\overline{t}$ (s)
This research method	423.27	94.16	517.43	0.1256
Nonfeature screening, binary method	903.75	286.37	1190.12	0.3217
Feature screening, binary method	392.56	106.12	498.68	0.1282
Nonfeature screening, weight allocation	883.24	230.69	1113.93	0.3012

accuracy performance and efficiency performance of the four methods. Figure 4(b) shows the results of these four methods on accuracy rate A, precision rate, recall rate, and F_1 value. It can be seen from the figure that the accuracy rate of the malicious detection method proposed in this research reached 92.17%, and the precision rate reached 94.68%. It is obviously better than the other three methods in each index. Replacing the binary method with the weight allocation method, replacing the malicious application detection method by obtaining the sensitive permission feature set can increase the fine-grained of malware detection and improve malware detection accuracy.

Table 2 shows the efficiency performance of the above four methods in training time, prediction time, total time, and average sample time. In the case of feature screening, the weight allocation method and the binary method are almost the same in each time index. When the sensitive permission feature set is not extracted, the redundant features are too much, and the measured time is significantly increased. This shows that the method proposed in this paper successfully reduces the dimension of digital features and improves malicious detection efficiency. Compared with the traditional method of direct detection of unfiltered permission features, the proposed method reduces the time by 60.94%.

To sum up, the detection method for malicious Android applications proposed in this paper based on permission sensitivity has an accuracy rate of 92.17%, 60.94% less than the traditional detection method.

5. Conclusion

In this paper, an Android malware detection method based on permission sensitivity is proposed. Based on the application category label and permission request information, the concepts of permission vector, permission matrix, and permission sensitivity are established, and the sensitive permission feature set corresponding to each category label is obtained by mathematical formula reasoning. Through the weight allocation method, each permission feature is given a weight value that measures the strength of distinguishing malicious and benign samples to form a digital feature. The classifier constructed by the random forest algorithm is used for malicious detection, and multiple sets of experiments are set for verification and comparison.

The experimental results show that the accuracy rate of the Android malware detection method can reach 92.17%, and the precision rate can reach 94.68%. Compared with the traditional detection method based on permission features, the time consumption of the malware detection method is reduced by 60.94%.

Data Availability

The data in this paper are divided into benign samples and malicious samples. The malicious samples were taken from the VirusShare and Drebin data sets. The benign samples are available from the corresponding author upon request.

Conflicts of Interest

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

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (grant no. 61601113) and the National Key R&D Program of China (grant nos. 2018YFB2202200 and 2018YFB2100403).

References

[1] T. Wu and Y. Yang, "Detecting android inter-app data leakage via compositional concolic walking," *Intelligent Automation & Soft Computing*, vol. 25, no. 4, pp. 755–766, 2019.

[2] R. Song, Y. Song, Q. Dong, A. Hu, and S. Gao, "Weblogger: stealing your personal pins via mobile web application," in *Proceedings of the 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6, Nanjing, China, October 2017.

[3] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM'11)*, pp. 15–26, Association for Computing Machinery, Chicago, IL, USA, August 2011.

[4] C. Yuan, S. Wei, Y. Wang, Y. You, and S. G. ZiLiang, "Android applications categorization using bayesian classification," in *Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 173–176, Chengdu, China, October 2016.

[5] A. Kutlay and K. Karaduzovic-Hadziabdic, "Static based classification of malicious software using machine learning methods," in *Proceedings of the Advanced Technologies, Systems, and Applications IV -Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies (IAT 2019)*, Sarajevo, Bosnia and Herzegovina, June 2019.

[6] F. Ahmad, A. N. Badrul, K. Ahmad et al., "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 6, pp. 712–736, 2018.

[7] W.-C. Wu and S.-H. Hung, "DroidDolphin: a dynamic Android malware detection framework using big data and machine learning," in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems (RACS'14)*, pp. 247–252, Association for Computing Machinery, Towson, MD, USA, October 2014.

[8] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, vol. 8, Article ID 96899, 2020.

[9] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proceedings of the IEEE 39th Annual Computer Software and Applications Conference*, pp. 422–433, Taichung, Taiwan, July 2015.

[10] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, March 2019.

[11] Y. Li, G. Xu, H. Xian, L. Rao, and J. Shi, "Novel android malware detection method based on multi-dimensional hybrid features extraction and analysis," *Intelligent Automation and Soft Computing*, vol. 25, no. 3, pp. 637–647, 2019.

[12] Z. Zhou, J. Qin, X. Xiang, Y. Tan, Q. Liu, and N. Xiong, "News text topic clustering optimized method based on tf-idf algorithm on spark," *Computers, Materials & Continua*, vol. 62, no. 1, pp. 217–231, 2020.

[13] L. Yan, Y. Zheng, and J. Cao, "Few-shot learning for short text classification," *Multimedia Tools and Applications*, vol. 77, no. 22, Article ID 29799, 2018.

[14] N. Chayangkoon and A. Srivihok, "Feature reduction of short text classification by using bag of words and word embedding," *International Journal of Control and Automation*, vol. 12, no. 2, pp. 1–16, 2019.

[15] D.-W. Kim, G.-Y. Shin, and M.-M. Han, "Analysis of feature importance and interpretation for malware classification," *Computers, Materials & Continua*, vol. 65, no. 3, pp. 1891–1904, 2020.

[16] "Virus share [EB/OL]," https://virusshare.com/.

[17] D. Arp, M. Spreitzenbarth, M. Hubner et al., "Drebin: effective and explainable detection of android malware in your pocket," *Ndss*, vol. 14, pp. 23–26, 2014.

[18] M. Anshori, I. F. Mar, and F. A. Bachtiar, "Comparison of machine learning methods for android malicious software classification based on system call," in *Proceedings of the International Conference on Sustainable Information Engineering and Technology (SIET)*, Lombok, Indonesia, September 2019.

[19] P. R. K. Varma, K. P. Raj, and K. V. S. Raju, "Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms," in *Proceedings of the International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 294–299, Palladam, India, February 2017.

[20] X. Qin, S. Peng, X. Yang, and Y.-D. Yao, "Deep learning based channel code recognition using TextCNN," in *Proceedings of the IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–5, Newark, NJ, USA, November 2019.