Hindawi
Security and Communication Networks
Volume 2020, Article ID 8841233, 11 pages
https://doi.org/10.1155/2020/8841233

Research Article

An Android Malware Detection Model Based on DT-SVM

Min Yang,[1] **Xingshu Chen**,[2] **Yonggang Luo**,[2] **and Hang Zhang**[3]

[1]*College of Cybersecurity, Sichuan University, Chengdu, China*
[2]*College of Cybersecurity and the Cybersecurity Research Institute, Sichuan University, Chengdu, China*
[3]*Technology and Engineering Group, Tencent, Shenzhen, China*

Correspondence should be addressed to Xingshu Chen; chenxsh@scu.edu.cn

Received 31 August 2020; Revised 8 October 2020; Accepted 26 November 2020; Published 15 December 2020

Academic Editor: Zhe-Li Liu

In order to improve the accuracy and efficiency of Android malware detection, an Android malware detection model based on decision tree (DT) with support vector machine (SVM) algorithm (DT-SVM) is proposed. Firstly, the original opcode, Dalvik opcode, is extracted by reversing Android software, and the eigenvector of the sample is generated by using the n-gram model. Then, a decision tree is generated via training the sample and updating decision nodes as SVM nodes from the bottom up according to the evaluation result of the test set in the decision path. The model effectively combines DT with SVM. Under the premise of maintaining a high-accuracy decision path, SVM is used to effectively reduce the overfitting problem in DT and thus improve the generalization ability, and maintain the superiority of SVM for the small sample training set. Finally, to test our approach, several simulation experiments are carried out, and the results demonstrate that the improved algorithm has better accuracy and higher speed as compared with other malware detection approaches.

1. Introduction

In recent years, mobile Internet has played a leading role in the evolvement of the Internet, and smartphones have become almost an indispensable tool in people's daily life. Smartphone penetration among adults in developed countries will reach 90 percent by the end of 2023, compared with 85 percent in 2018, and global smartphone sales will reach 1.85 billion units, 19% increase over 2018 [1]. According to [2], worldwide sales of smartphones to end users are on track to reach 1.57 billion units in 2020, an increase of 3% year over year. Although the market sales of smartphone went through a slight declination in 2019, Gartner forecasts that sales of 5G mobile phones will total 221 million units in 2020, and more than double in 2021, to 489 million units; there is no doubt that the gradual maturity of 5G technology will also push the demand of smartphones rise considerably.

Currently, the common operating systems of smartphone terminals include iOS, Android, and Windows Phone, among which Android, in particular, became the dominating operating system with the highest market share on a global scale because of its open-source nature, which gives users and developers the flexibility to customize basic functionality [3]. According to survey data released by Gartner, the share of the Android system in 2017 was as high as 85.9% [4]. However, the increasing popularity of Android is also accompanied by the proliferation of malware. In 2018, 360 Internet Security Center intercepted about 4.342 million new malicious samples on the mobile terminal, with an average of about 12,000 new ones added every day. The new malware types are mainly tariff consuming, accounting for about 63.2%, followed by privacy theft 33.7%, malicious deduction 1.6%, rogue behavior 1.2%, and remote control 0.3% [5]. The terminal application endangers the users' interests by allowing unauthorized access to privacy-sensitive information, rooting devices, monitoring their daily behaviors, etc. [6]. The amount of malware continues to grow at a faster rate each year and poses a serious security threat, antivirus vendors detect thousands of new malware samples daily, and there is still no end in sight [7]. In particular, with the gradual maturity of 5G technology, which marks the arrival of the era of intelligent networking and industrial Internet, the Internet of everything will lead to more lethal and wider harm caused by malware, and hence, malware detection has been and will be a critical topic in computer security.

In this study, we develop a DT with the SVM algorithm (DT-SVM) for improving the detection efficiency and accuracy of malware on the Android platform. The major contributions of this work can be summarized as follows:

(i) We develop an advanced machine learning algorithm, which firstly extracts the opcode of samples; then, n-gram is utilized to vectorize and train the sample to generate the decision tree; and, finally, the nodes with high error are updated from the bottom up as SVM nodes. The algorithm combines the advantages of DT and SVM; on the premise that high accuracy is maintained, the SVM node is employed to reduce the overfitting problem caused by DT. Therefore, the algorithm takes full advantage of the SVM in a small sample set and has a better classification effect than merely using DT or SVM separately.

(ii) We design an Android malware detection framework based on DT-SVM algorithm. The framework is trained based on the improved learning algorithm with the malicious and benign applications utilized, and feature vectors of these applications are generated by Android reverse engineering, feature engineering, and n-gram, which are used as the input of the proposed algorithm for malicious detection. In this way, users can employ our proposed framework to distinguish whether the application is malicious or benign before installation; thus, the Android platform security issues can be greatly improved.

(iii) We verify the effectiveness of our advanced algorithm based on real-word benign applications and malware, perform malicious detection on the same dataset of the proposed algorithm with the shallow learning algorithms DT and SVM and the deep learning algorithms CNN and LSTM, and use four evaluation metrics (Precsion, ACC, Recall, *F*1) as well as time consumption to measure the performance of the algorithm. The results demonstrate that our proposed algorithm performs better than SVM, DT, and LSTM almost in all metrics and performs better than CNN in some metrics. All the four metrics, that is, Precision, Recall rate, ACC, and F1, increase by nearly 0.01% compared with SVM, while the time consumption reduces to one-tenth, as well as increasing by nearly 0.03% separately compared to DT with time consumption not changed much. Compared with CNN, although ACC and F1 are lower, Precision and Recall are higher; furthermore, our algorithm takes less time, and the implementation process is much simpler. In terms of LSTM, our method performs better than it in all metrics.

The remainder of this paper is organized as follows. Section 2 states some current work of Android malware detection. Section 3 depicts the related methodology. Section 4 describes the proposed classification algorithm. Section 5 illustrates the Android malware detection framework and explains the specific process of applying the proposed algorithm to the detection of malicious applications. Section 6 verifies the effectiveness of the advanced algorithm based on Android applications. Section 7 concludes the paper and points out the main limitations and future directions.

2. Related Work

There have been a lot of achievements in terms of detecting malware on the Android platform, which can be divided into two analysis approaches, that is, static analysis and dynamic analysis [8]. Static analysis is the process of analyzing the code or binary without executing it. Dynamic analysis is the process of studying traces of the malware (API, system calls, permission, etc.) through running the sample in a controlled and isolated environment [9]. Traditionally, malware detectors have been built on handmade detection patterns that are not usually applicable to new instances of malware; however, the increasing number and diversity of these applications make traditional defenses largely ineffective; Android smartphones often fail to protect themselves from new malware [10]. Owing to the emergence of machine learning technology, which can potentially detect never-before-seen attacks or variants of known malware with its strong generalization and prediction ability, machine learning-based methods are increasingly applied to Android malware detection by researchers, and the improvement of classic algorithms has always been the tireless work of scholars. The shallow learning model and the deep learning model are the two main types of machine learning techniques [11]. The shallow learning model usually includes SVM, DT, and k-means as well as k-nearest neighbor (KNN) algorithms, etc. [12]. Reference [13] improved the accuracy of the classifier by using machine learning to extract features from the system call of Android malware. Due to the high feature dimension in Dalvik opcode-based detection, [14, 15] utilized two strategies of probability statistics and feature extraction to effectively reduce the dimensionality of extracted features, and the linear SVM was employed for classification, and therefore, the inspection efficiency was improved. Based on the characteristics of permission information and Intent information in AndroidManifest.xml file, a random forest improvement algorithm based on weighted voting was proposed in [16], and the inability to distinguish strong and weak classifiers was solved. Nancy and Sharma [17] compared the network traffic of malware with that of benign applications to find out the characteristics that distinguish the two types of traffic and built a DT classifier to detect normal and malicious applications from the test dataset. The results showed that the network traffic analysis method was efficient in detecting Android malware, with an accuracy rate of more than 90%. Nevertheless, most of the work mentioned above has not achieved decent performance. Recently, Android malware researchers have also been exploring deep learning classifiers for malware analysis to increase detection accuracy [18]. Cui et al. [19] took the advantage of the performance of deep learning in image recognition; the malicious detection code was converted into a grayscale image as the input of CNN under the condition of the fixed image size, which was not realistic in a real scenario. Therefore, this method suffered from fluctuating in performance when processing different sizes of images. To improve the accuracy of malware detection and reduce the training time, Wang et al. [20] proposed a hybrid model based on deep autoencoder (DAE) and convolutional neural network (CNN); the experiments demonstrated a significant improvement compared with traditional machine learning methods in Android malware detection. Wang et al. [21] ranked the permissions w.r.t. their risk to the Android system and evaluated the feasibility of using permission requests for malapp detection with different subsets of risky permissions and classification algorithms; the detection rate can achieve 94.62%. Furthermore, the author considered the issue of user privacy information leakage in literature [22] and implemented a framework called 'Alde' to detect the users in-app actions collected by analytics libraries; experimental results show that some apps indeed leak users personal information through analytics libraries. Lei et al. [23] adopted more advanced features than the API event behavior model as a data source, using different behavior patterns of events and the semantic relationships between events to detect malicious software. This method can effectively solve the problem of confusion deformation. However, the results of the experiment performed quite well only in the malware dataset provided in 2013. As the complexity of the malware increased, the detection ability declined.

In summary, it can be concluded that there are two ways to improve the detection accuracy and efficiency of Android malware; the first is through optimization of feature selection and detection model, and the second is to optimize classification algorithms. We mainly focus on the latter and improve the classic classification algorithm in this study. SVM is simple and can achieve high classification accuracy. However, it is merely suitable for small samples; if the sample set is large, it will consume a lot of time and have a high false positive rate. DT is easy to overfit, leading to weak generalization ability of prediction results. To overcome these limitations, our work proposes an advanced learning algorithm based on static features and combines the advantages of SVM and DT algorithm, and the experimental results are quite good. In the next section, we explain the methodology.

3. Methodology

3.1. N-Gram. N-gram model is derived from Natural Language Processing (NLP), commonly used in large-scale continuous speech recognition, which believes that the appearance of the N_{th} word must be related to the first N – 1 words, but not to other words. Hence, the probability of the entire sentence should be equal to the probability product of the occurrence of each word. N-gram can also be used in malware detection. As early as 2008, Moskovitch et al. [24] proposed the opcode n-gram scheme and achieved good detection results.

3.2. Support Vector Machine (SVM). SVM [25] is a two-class model whose basic model is a linear classifier that defines the interval maximization in the eigenspace. Meanwhile, it can also solve the nonlinear problem employing kernel trick [26]. The learning strategy of SVM is to maximize the interval, which can be formalized as a problem of solving convex quadratic programming, also called the maximum edge algorithm, whose advantage lies in strong generalization ability, which can solve the issues of nonlinear, small samples, high dimension, etc. Taking the linear separable SVM as an example, the principle of SVM is to search for a separable hyperplane in given eigenspace and then divide the sample space into two categories, one is a positive class and the other is a negative class, corresponding to two different categories of samples. The hyperplane H in the support vector machine can be represented by the equation of $w \cdot x + b = 0$, where w is the normal vector and b is the intercept, as shown in Figure 1.

When the training samples are linearly separable, there are many straight lines that can correctly classify the two types of samples, and SVM is to find the line that can correctly divide them with the largest interval. SVM also supports nonlinear problem classification, whose main character is the utilization of kernel trick, the basic idea behind which is to match the input space to an eigenspace, so that its hypersurface model in the input space corresponds to the hyperplane model in the eigenspace through a nonlinear transformation. The radial basis function (RBF) is one of the commonly used kernel functions.

Definition 1. Gaussian kernel function

$$K(x, z) = \exp\left(-\frac{\|x - z\|_2^2}{2\sigma^2}\right). \tag{1}$$

Here, $\|x - z\|_2^2$ is the square Euclidean distance of two eigenvectors, and σ is a free parameter.

3.3. Decision Tree. Decision tree [27] is a basic classification and regression method, which classifies samples into a tree structure, represents the process of classifying samples based on features in classification problems, and can also be considered as a collection of if-then rules. DT is widely used because of its intuitive feature description, high classification accuracy, and simple implementation [28]. The learning process of DT is to find a mapping relationship between the object attribute and the object value, enabling it to generalize a set of classification rules represented by tree structure from random samples. The decision path of DT has important properties: mutual exclusion and completeness; that is, each instance is covered by the one and the only one path. The learning algorithm of DT includes feature selection, decision tree generation, and pruning process. The widely used generation algorithms of DT are ID3, C4.5, and CART. The Gini index is used for optimal feature selection in CART algorithm.

Definition 2. Gini index

In the classification problem, suppose that there are K classes and the probability that the sample belongs to the k_{th} class is p_k; then, the Gini index of the probability distribution is defined as

$$\text{Gini}(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2. \quad (2)$$

In the dichotomy problem, the Gini index of the sample set D is expressed as

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2. \quad (3)$$

Here, $|C_k|$ represents the number of samples in category k, and $|D|$ represents the total number of samples. The Gini index indicates the uncertainty of the sample set. The larger the Gini index, the greater the uncertainty of the sample set.

4. Decision Tree with SVM Algorithm (DT-SVM)

To overcome the problem of overfitting and weak generalization ability in DT algorithm, DT-SVM is proposed. SVM is embedded into DT for node optimization, which not only ensures the high accuracy of the decision path and improves the generalization ability of DT, but also takes advantage of SVM in small sample training. DT-SVM aims to create a decision model as shown in Figure 2. The process of the algorithm is to generate a decision tree based on the sample set and then update the decision node from the bottom up.

DT is a supervised learning algorithm. The sample set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ is divided into the training set and the test set, denoted by TrainSet and TestSet.

Definition 3. Assume that the decision tree is as shown in Figure 3, where the leaf nodes are instance sets, represented by $S = \{d_1, d_2, \ldots, d_n\}$, where n is the number of leaf nodes. The nonleaf node is a feature set and is denoted by $C = \{c_1, c_2, \ldots, c_n\}$. Each leaf node corresponds to a decision path, the decision path corresponding to the leaf node j is defined as $\text{dp}_j = \{c_1, c_k, \ldots, d_j\}$, and $h = \text{len}(\text{dp}_j)$ indicates the depth of the path. The details of our suggested DT-SVM algorithm for Android malware detection are presented in Steps 1 to 8.

According to the algorithm process, assume that the initial decision tree is shown in Figure 4 and the DT-SVM tree generated by the algorithm is shown in Figure 5.

The algorithm has a good performance in the example illustrated by Figure 6, in which the sample set cannot be effectively segmented, adopting DT and SVM algorithm separately, but the DT-SVM algorithm can preserve the high precision decision path and optimize nodes with low precision as SVM nodes.

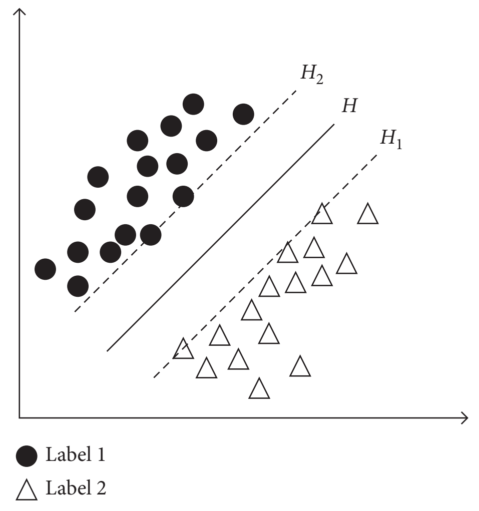

Figure 1: The hyperplane of SVM.

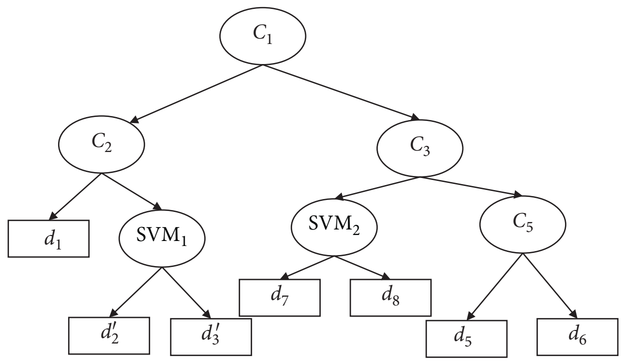

Figure 2: Decision tree with SVM nodes.

5. DT-SVM-Based Malware Detection Framework

5.1. Model Overview. The DT-SVM-based malware detection framework is shown in Figure 7. The framework consists of four modules, that is, instruction extraction, feature engineering, classifier training, and result evaluation.

5.2. Sample Instruction Extraction. Firstly, those samples are labeled as two categories, positive and negative. Then, opcode extraction is performed for each apk. After apk decompression, the core classes.dex file of the app will be obtained. The classes.dex file is the executable file of the Android system, which contains all operation instructions and data required by the runtime. The dex file can be parsed by 010 Editor, and its structure is shown in Figure 8. The Methods structure contains all the methods of the app, represented by the DexMethod structure.

```
struct DexMethod{
  /* Index pointing to the list of DexMethodId*/
  u4 methodIdx;
  u4 accessFlags;
  /* offset to the DexCode structure  * /}
```

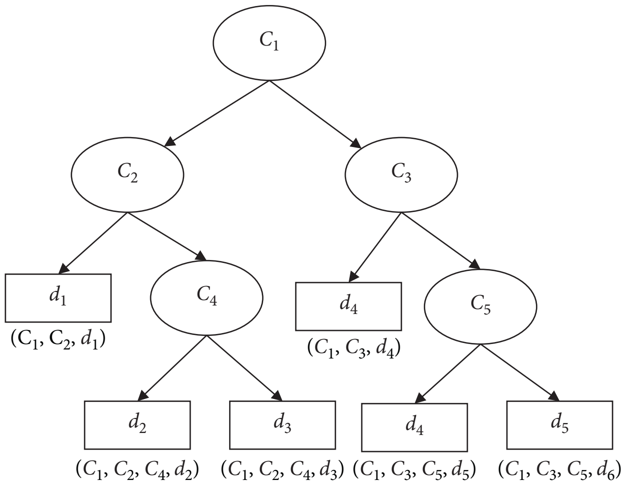

FIGURE 3: Traditional decision tree labeled with decision path.

Step 1. According to the training set *TrainSet*, the Gini index is used for feature selection and prepruning, and the decision tree T is constructed.
Step 2. Use the test set *TestSet* to evaluate the decision tree and calculate the *Precision* of each decision path p_i, then constitute the decision object do = $(\mathrm{dp}_i, p_i, h_i)$, and set the decision path accuracy threshold *Th*.
Step 3. Initialize the queue Q = { }, sort the decision objects generated in step 2 in descending order according to the path depth h of the decision path dp, and sequentially add them to the queue *Q*.
Step 4. Determine if the queue is empty. If it is, the algorithm ends. Otherwise, go to step 5.
Step 5. Fetch the element $q = (\mathrm{dp}, p, h)$ from the queue, and compare the decision path Precision rate p with the preset threshold *Th*. If it is less than the threshold, go to step 6; otherwise, retain the decision path and go to step 4.
Step 6. Determine whether the sibling node of q is a leaf node. If it is, go to step 7; otherwise, go to step 8.
Step 7. Determine whether the Precision of the path of q's sibling nodes is lower than the threshold *Th*. If it is, all the samples passing through the two decision paths (both path of q and q's siblings) are taken as a training set, which is trained with the SVM model and then merged and updated as SVM nodes; thereafter, the process proceeds to step 4.
Step 8. Take out all the training sets of the path of p, train them with the SVM model, and update them to SVM nodes. Then, go to step 4 and continue to traverse so as to update nodes.

ALGORITHM 1: The detailed procedure of DT-SVM.

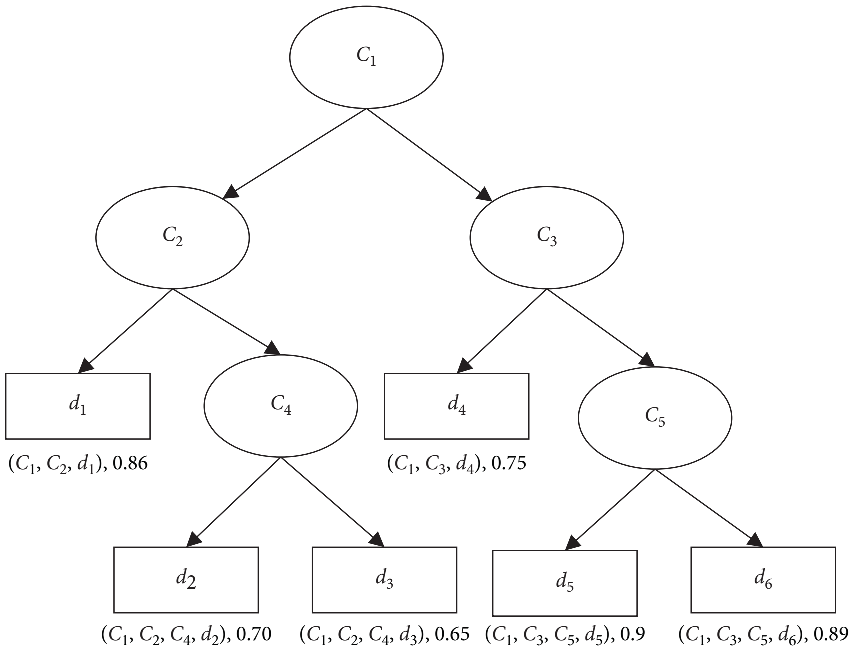
C_1
C_2
C_3
d_1
C_4
d_4
C_5
(C_1, C_2, d_1), 0.86
(C_1, C_3, d_4), 0.75
d_2
d_3
d_5
d_6
(C_1, C_2, C_4, d_2), 0.70
(C_1, C_2, C_4, d_3), 0.65
(C_1, C_3, C_5, d_5), 0.9
(C_1, C_3, C_5, d_6), 0.89

FIGURE 4: An instance with traditional decision tree.

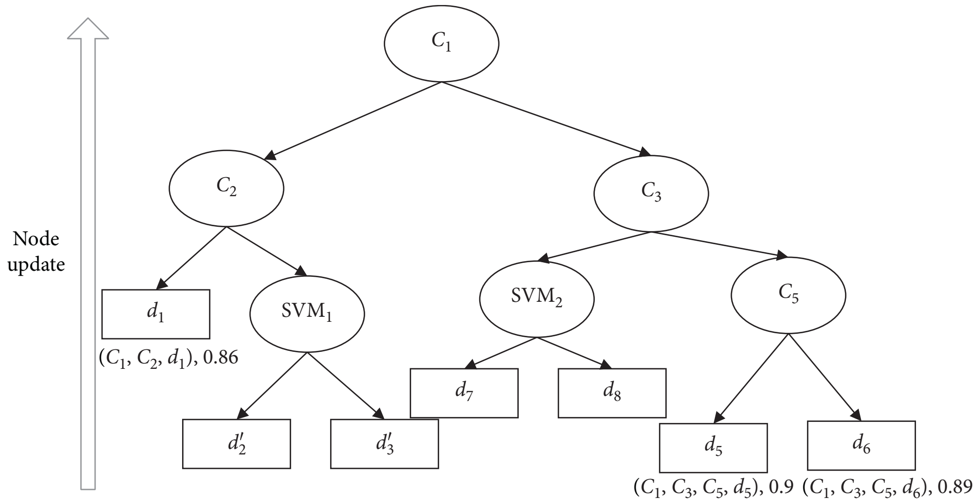

Figure 5: Updating decision tree with SVM nodes from the bottom up.

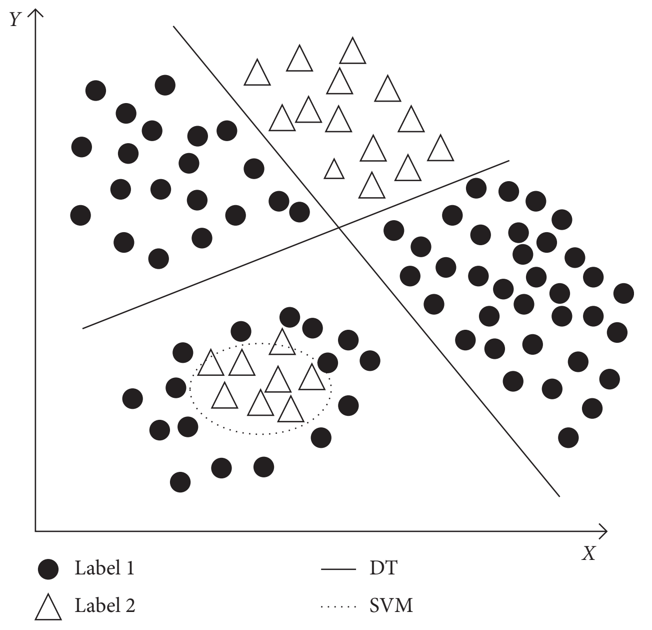

Figure 6: Classification of samples with DT-SVM algorithm.

```
 u4 codeOff;
 }
struct DexCode{
 /* the number of used registers */
 u2 registersSize;
 /* the number of parameters */
 u2 insSize;
 /* the number of used registers when calling other
methods */
 u2 outSize;
/* the number of try and catch */
 u2 triesSize;
 /* offset to debug information */
 u4 debugInfoOff;
 /* the number of Instruction Set */
 u4 insnsSize;
 /* Instruction Set */
 u2 insns [1];
}
```

In this structure, the last field insns[1] contains all the instruction sets of the method, namely, the corresponding Dalvik opcode. By going through all the methods, all opcode instructions can be fetched according to the Dalvik opcode instruction list in Table 1.

5.3. Feature Engineering. Since there are more than 200 Dalvik instructions, if all of them are directly input into the n-gram model, the feature dimension will be too high. In this paper, first of all, the Dalvik instruction sets are simply

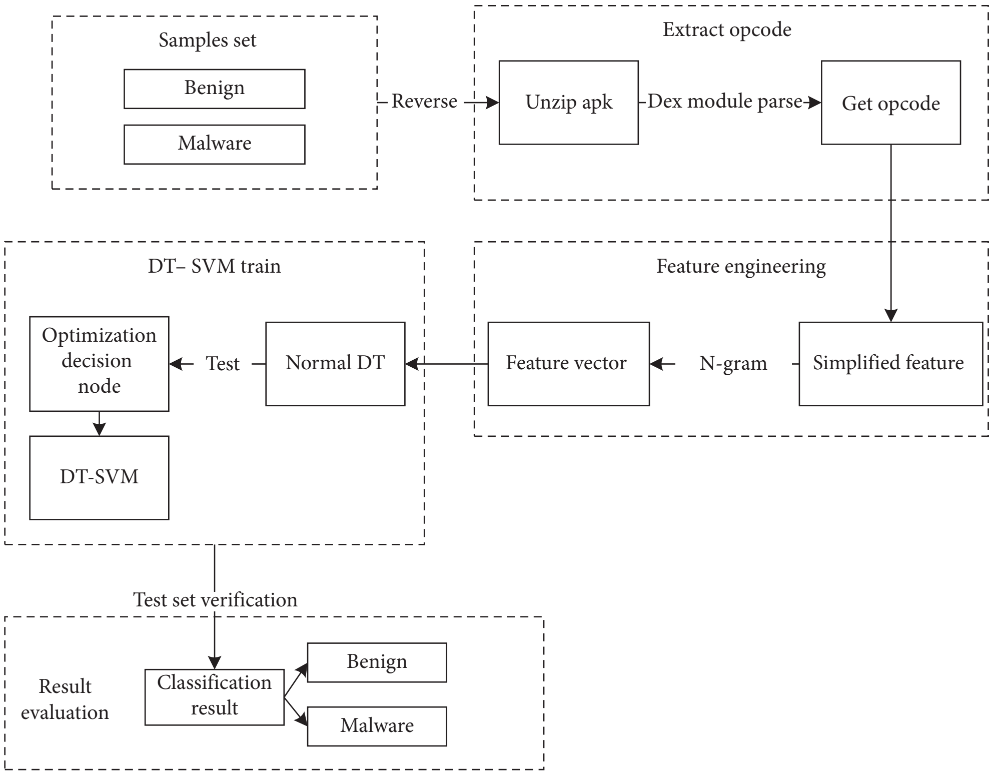

Figure 7: Android malware detection model based on DT-SVM algorithm.

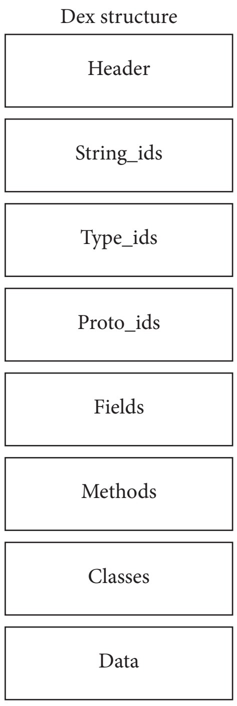

Figure 8: Android dex file structure.

classified; then, irrelevant instructions are removed; and, finally, only eight types are left. The opcode and its corresponding identifier are shown in Table 2.

Table 1: Dalvik opcode.

Opcode (hex)	Opcode name	Length
00	nop	2
01	move vx, vy	2
02	move/from16 vx, vy	4
03	move/16	6
04	move-wide	2
05	move-wide/from16 vx, vy	4
06	move-wide/16	6
07	move-object vx, vy	2
08	move-object/from16 vx, vy	4
09	move-object/16	6
0A	move-result vx	2
0B	move-result-wide vx	2
0C	move-result-object vx	2
0E	return-void	2
0F	return vx	2
10	return-wide vx	2
...	...	...

After simplifying the Dalvik instruction sets, all of them can be input to the n-gram model to generate sample eigenspace. The extracted opcode for each sample in Section 5.2 is mapped to the identifier, and the n-gram vector is constructed. Assuming that the Dalvik instruction is $\{G, P, V, I, J, R, M, C\}$, when $N = 3$, the 3-gram vector is $[\{GPV\}, \{PVI\}, \{VIJ\}, \{JRM\}, \{RMC\}]$.

After the n-gram model of samples is obtained, the n-gram types are counted. If a feature appears in the sample,

Table 2: Feature simplification mapping.

Identifier	Description	Opcode
G	Fetching data	aget—iget—sget—aget-wide—aget-object—aget-boolean—aget-byte—aget-char
P	Storing data	aput—iput—sput—aput-wide—aput-object—aput-boolean—aput-byte—aput-char
V	Method call	invoke-virtual—invoke-super—invoke-direct—invoke-static
I	Judgement	if-eq—if-ne—if-lt—if-ge—if-gt—if-le—if-eqz—if-nez—if-ltz—if-ltz—if-gez—if-gtz—if-lez
J	Goto	goto—goto/16—goto/32
R	Return	return—return-void—return-wide—return-object
M	Move	move—move-wide—move-object—move-result—move-exception
C	Compare	cmpl-float— cmpg-float— cmpl-double— cmpg-double—cmp-long

the value of the feature is set to 1; otherwise, it is set to 0; the feature vector of the sample is finally obtained.

5.4. Evaluation Metrics. Four metrics are employed to verify the performance of our proposed algorithm, namely, Precision, Recall, classification accuracy ACC, and F1 value, which are broadly used in machine learning. The Precision can be denoted as

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

where TP (true positive) indicates the number of Android malware samples which are correctly detected and FP (false positive) indicates the number of benign applications that are wrongly detected as Android malware [29]. In this study, the Precision refers to the ratio of the identified malicious samples to the real malicious samples. The Recall can be formulated as

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

where FN (false negative) indicates the number of Android malware samples that are not detected (predicted as benign applications) [29]. In this study, Recall reflects the proportion of malicious samples identified in the real malicious sample. The ACC can be formulated as

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

where TN (true negative) represents the number of benign applications that are correctly classified and ACC is an overall evaluation of the classifier, representing the proportion of the total number of the applications that are correctly classified whether as benign or malicious. The higher the ACC is, the better the performance will be. F1 is the harmonic mean between the Precision and Recall; it can be denoted as

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}. \tag{7}$$

6. Experimental Simulation

6.1. Datasets. In the experimental simulation environment, the malicious sample set was obtained from the malicious sample database in the Drebin project of the University of Gottingen, Germany [30], in which the malware samples are 5560 in total, and the time range was from August 2010 to October 2012. An overview of the top 20 malware families in the dataset is provided in Table 3, including several families that are currently actively distributed in application markets. There are 4414 benign samples, and the benign samples were randomly selected from the applications, which were downloaded from the Google Play app store in order of ranking through the crawler module. The tools used in the experiment include unzip, dexParser, scikit-learn, etc. Scikit-learn is an excellent Python programming machine learning library, which has a variety of classification, regression, and clustering algorithms, including support vector machine, random forest, and gradient enhancement.

Table 3: Top malware families in our dataset.

ID	Family	Number
A	FakeInstaller	925
B	DroidKungFu	667
C	Plankton	625
D	OpFake	613
E	GingerMaster	339
F	BaseBridge	330
G	Iconosys	152
H	$K_{\min}$	147
I	FakeDoc	132
J	Geinimi	92
K	Adrd	91
L	DroidDream	81
M	Linux/Lotoor	70
N	GoldDream	69
O	MobileTx	69
P	FakeRun	61
Q	SendPay	59
R	Gappusin	58
S	Imlog	43
T	SMSreg	41

6.2. Experimental Procedure. The sample set was divided into a training set, a pseudo test set, and a test set in the ratio of 6 : 2:2. The training set feature vector was input into the DT-SVM model for training. The pseudo test set was used to update the decision node and obtain the DT-

SVM tree. Finally, the test set was employed to evaluate the performance of the classifier.

The experiment used 1638 malicious samples and 1324 normal samples. 60% of the training sets and 20% of the pseudo test sets were used to generate the DT-SVM model, and then the remaining 20% were used to evaluate the classifier performance. 3-gram was used for feature selection. Since different sampling will affect the classification results, the experiment will perform 10-fold cross-validation.

In order to ensure that the decision leaf node has sufficient sample capacity for SVM training, the decision tree needs to be prepruned. In the experiment, the minimum sample number of the leaf node min_samples_leaf is 40, the maximum depth of the decision tree max_depth is 5, and the Precision threshold is set to 0.9. The decision tree path below the threshold is shown in Table 4, where the field of 'Path matrix' is the binary representation of decision path. The encoding process is to sort all nodes of a decision tree from left to right and from top to bottom, and then map them to a multidimensional vector. The position of this multidimensional vector represents the sort of decision tree node, and the value represents whether the decision path contains this node. If it is 1, the node is included; otherwise, it is not included.

For these decision paths with higher error, the samples under each path are taken out for SVM training to generate SVM nodes. The Gaussian kernel function is used to process the feature space during training. At this time, there are two essential parameters that need to be adjusted, namely, the C (Penalty factor) and gamma (RBF kernel width). In general, a larger C leads to higher tolerance, but fewer errors, so as to eliminate overfitting. Otherwise, it is easy to result in underfitting. Gamma is a parameter of the Gaussian kernel function. The larger the gamma is, the less the support vector is, and the simpler the model is.

After training, the parameters of each SVM node are shown in Table 5.

6.3. Experimental Results

6.3.1. Scenario I: The Impact of Different N-Gram Types on the Classifier. DT and SVM classifiers were trained separately applying different n-gram models, and the predictive Precision results are shown in Table 6.

The results show that DT and SVM can get good evaluation results on the basis of 3-gram and 4-gram, demonstrating the feasibility of the modeling method. When $n > 3$, the Precision of DT only increases by 0.7%; SVM increases by 2%, but it consumes a lot of time. SVM takes 1002.23 seconds under 4-gram and 113.65 seconds under 3-gram, so $n = 3$ gives the best performance for sample vectorization.

6.3.2. Scenario II: Results Comparison with Shallow Learning Algorithm. The sample was vectorized based on 3-gram, and Table 7 demonstrates a comparison of the proposed algorithm with SVM and DT for Android malware detection.

The results show that the Precision, ACC, Recall, and $F1$ of the DT-SVM algorithm are apparently higher than traditional DT and slightly higher than SVM. In terms of efficiency, SVM takes the longest time, while DT-SVM is trained by DT first, and then the small sample is trained by the SVM node. Hence, the time dramatically reduces compared with SVM, albeit a little longer than DT.

6.3.3. Scenario III: Results Comparison with Deep Learning Algorithm. We also compared the CNN [31] and LSTM [32] using the same sample set for training. The results show that ACC and $F1$ of CNN are relatively high, but other metrics are lower than our proposed model, which means that there would be a lot of false positives of CNN. In addition, CNN is time consuming and requires high machine configuration. The performance of the LSTM model for malicious detection of Android is not so good as that of DT-SVM algorithm, and the time consumption is 117s higher than that of our algorithm. The results are detailed in Table 8.

6.3.4. Scenario IV: Comparison of DT-SVM Results with Different Sample Sizes. We randomly select 507 samples from the 2962-sample set for experiment. The effects of different sample sizes on DT-SVM classifier are shown in Table 9 .

The experimental results show that the sample size has a certain influence on the detection effect. The number of samples increases, and Precison, ACC, Recall, and $F1$ increase by 0.03. Hence, we can conclude that the larger the sample size is, the better the overall performance will be.

6.4. Analysis. Decision tree is a prediction model, which represents a mapping relationship between object attributes and object values. Its branches classify objects of this type based on attributes. It is a decision tool using a decision model, which can help determine a strategy most likely to achieve the goal. DT is easy to understand and implement, the advantage of which lies in its ability to make accurate and feasible predictions for large data sources in a short time. The basic principle of DT-SVM is to first extract some high-accuracy decisions through DT model and quickly find the strong correlation between the results and the attributes, and then the kernel technique of SVM is used to solve nonlinear prediction for some weakly correlated samples and at the same time give full play to the advantages of SVM in small sample prediction. Hence, the prediction accuracy of the samples is largely improved through the combination of DT and SVM.

The time complexity of DT is $O(n \log n)$, and SVM is $O(n^3)$. However, DT-SVM first generates a decision tree, selects the optimal path, and then uses SVM for training for the remaining samples, so the time complexity is $O(n \log n) + O(m^3)$, where n is the total number of samples and m is the number of samples that cannot be distinguished with high accuracy after training the sample using the decision tree; $m \ll n$; thus, the value falls in the interval $(O(n \log n), O(n^3))$. In this experiment, decision tree was

Table 4: Decision tree path with Precision.

Path ID	Decision path	Path matrix	Precision
1	$(C_{296}, C_9, C_{313}, C_{304}, C_{308}, d_7)$	1100000110001010000000000	0.737
2	$(C_{296}, C_9, C_{120}, d_1)$	1111000000000000000000000	0.571
3	$(C_{296}, C_9, C_{313}, d_8)$	1100000100000001000000000	0.590
4	$(C_{296}, C_{307}, C_{223}, d_{10})$	1000000000000000101100000	0.685
5	(C_{296}, C_{307}, d_9)	1000000000000000110000000	0.850

Table 5: SVM node parameters.

SVM node ID	C	gamma
1	7	0.03
2	7	0.003
3	1	0.04
4	5	0.04
5	5	0.04

Table 6: The results of scenario I.

N-gram	DT	SVM
2-gram	0.79	0.76
3-gram	0.92	0.95
4-gram	0.94	0.97

Table 7: The results of scenarios II.

Classifier	Precision	ACC	Recall	F1	Time consumption
DT	0.92	0.93	0.93	0.93	8.01s
SVM	0.96	0.96	0.94	0.95	105.79s
DT-SVM	0.96	0.96	0.96	0.96	18.9s

Table 8: The results of scenario III.

Classifier	Precision	ACC	Recall	F1	Time consumption
LSTM	0.893	0.938	0.556	-	117.8s
CNN	0.944	1	0.944	0.971	357.24s
DT-SVM	0.96	0.96	0.96	0.96	18.9s

Table 9: The results of scenario IV.

Samples size	Precision	ACC	Recall	F1
507	0.93	0.93	0.94	0.93
2962	0.96	0.96	0.96	0.96

first used to train samples, and it can be found from Table 4 that the Precision of paths 1, 2, 3, 4, 5 is low, indicating that DT cannot accurately separate positive and negative samples. Taking path 2 $(C_{296}, C_9, C_{120}, d_1)$ as an example, by mapping and restoring, the opcode sequence corresponding to path 2 is JRG, GPP, PCG, where it is observed that JRG is a jump return to obtain data sequence, GPP is a data acquisition and storage sequence, and PCG is a data dump sequence. These sequences are often used for both positive and negative samples; therefore, merely using DT cannot distinguish them effectively (the accuracy is only 57.1%). Based on this, this paper trains these undifferentiated samples using SVM, and the Precision reaches as high as 96%. In summary, the proposed algorithm improves detection accuracy, while the time consumption is relatively low.

7. Conclusion and Future Work

Taking the sample Dalvik opcode as the research object, the n-gram model is utilized to generate the sample eigenvector, and DT-SVM is proposed. Based on the original DT, the proposed algorithm uses SVM to update the decision nodes from the bottom up. The advantages of DT and SVM can be combined through DT-SVM, and the disadvantages of overfitting of DT and low accuracy of SVM for large samples are overcome. Finally, the superiority of the algorithm is demonstrated by simulation experiments, and good results are obtained in Android malicious apps detection.

However, in addition to the above advantages, there are some limitations to our study. This paper only performs static analysis on the sample; if the sample is hardened or confused, the unzip file will no longer be the sample's classes.dex, but the hardened executable file. The Dalvik opcode will be virtualized, and all instructions will be executed by a hardened virtual machine. At this time, opcode will no longer correspond to the Dalvik instruction list, and only the dynamic behavior analysis method can be used for malicious code detection. In addition, the proposed DT-SVM algorithm can still be improved by, for example, using the random forest to further improve the classification ability of DT-SVM and extending DT-SVM algorithm to the multiclassification decision model.

Data Availability

The data in this paper are divided into benign samples and malicious samples. The malicious sample data that support the findings of this study are available but restrictions apply to the availability of these data, which were used under license for the current study, and so they are not publicly available. These data are however available from the corresponding author upon reasonable request and with permission of the Drebin project of the University of Gottingen, Germany. The benign sample data generated and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] D. Insights, “Tech Trends 2019 beyond the Digital Frontier,” Tech. Rep., Deloitte, London, UK, 2019, https://www.innovation4.cn/library/r37176.

[2] U. K. Egham, “Gartner says worldwide smartphone sales willgrow 3Gartner, UK,” Stamford, CT, USA,2020, Tech. Rep., Gartner, https://www.gartner.com.

[3] R. Mente and A. Bagadi, “Android application security,” *Advances in Computational Sciences and Technology*, vol. 10, pp. 1207–1210, 05 2017.

[4] U. K. Egham, “Gartner says worldwide sales of smartphones recorded first ever decline during the fourth quarter of 2017,” Stamford, CT, USA, Tech. Rep. Gartner, https://www.gartner.com/newsroom/id/3876865.

[5] I. S. Center, “Android malware special report in 2018,” 360 Fenghuo Laboratory, Beijing, China, Tech. Rep., 2019, https://zt.360.cn/1101061855.php?dtidŁ1101061451&didŁ610100815.

[6] S. Y. Yerima and S. Sezer, “Droidfusion: A novel multilevel classifier fusion approach for android malware detection,” *IEEE Transactions on Cybernetics*, vol. 99, pp. 1–14, 2018.

[7] A. Demontis, M. Melis, B. Biggio et al., “Yes, machine learning can be more secure! a case study on android malware detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2017.

[8] N. Milosevic, A. Dehghantanha, and K.-K. R. Choo, “Machine learning aided android malware classification,” *Computers & Electrical Engineering*, vol. 61, pp. 266–274, 2017.

[9] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, “Malware dynamic analysis evasion techniques: a survey,” 2018, http://arxiv.org/abs/1811.01190.

[10] B. Biggio and F. Roli, “Wild patterns: ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[11] W. Zhong and F. Gu, “A multi-level deep learning system for malware detection,” *Expert Systems With Applications*, vol. 133, pp. 151–162, 2019.

[12] Z.-U. Rehman, S. N. Khan, K. Muhammad et al., “Machine learning-assisted signature and heuristic-based detection of malwares in android devices,” *Computers & Electrical Engineering*, vol. 69, pp. 828–841, 2018.

[13] 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.

[14] Y. Zhang and C. Yin, “Android malware detection based on svm,” *Journal of Shandong University (Engineering Science)*, vol. 47, no. 1, pp. 42–47, 2017.

[15] X. Liu, J. Weng, Y. Zhang, B. Feng, and J. Weng, “Android malware detection based on apk signature information feedback,” *Journal on Communications*, vol. 38, no. 5, pp. 190–198, 2017.

[16] H. Yang and J. Xu, “Android malware detection based on improved random forest,” *Journal on Communications*, vol. 38, no. 4, pp. 8–16, 2017.

[17] D. Nancy and D. Sharma, “Android malware detection using decision trees and network traffic,” *International Journal of Computer Science and Information Technologies*, vol. 7, no. 4, pp. 1970–1974, 2016.

[18] A. Mohammed K, Y. Suleiman, and S. Sezer, “Dl-droid: deep learning based android malware detection using real devices,” *Computers & Security*, vol. 89, 2019.

[19] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, “Detection of malicious code variants based on deep learning,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[20] W. Wang, M. Zhao, and J. Wang, “Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2018.

[21] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, “Exploring permission-induced risk in android applications for malicious application detection,” *IEEE Transactions on Information Forensics & Security*, vol. 9, no. 11, pp. 1869–1882, 2017.

[22] X. Liu, J. Liu, S. Zhu, W. Wang, and X. Zhang, “Privacy risk analysis and mitigation of analytics libraries in the android ecosystem,” *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1184–1199, 2020.

[23] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, “Evedroid: event-aware android malware detection against model degrading for iot devices,” *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[24] R. Moskovitch, C. Feher, N. Tzachar et al., “Unknown malcode detection using opcode representation,” in *Proceedings of the European Conference on Intelligence and Security Informatics*, pp. 204–215, Esbjerg, Denmark, December 2008.

[25] C.-C. Chang and C.-J. Lin, “Libsvm: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[26] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, Pittsburgh, PA, USA, July 1992.

[27] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[28] “Application of machine learning in cyberspace security research,” *Chinese Journal of Computers*, vol. 41, no. 9, 2018.

[29] A. Mahindru and P. Singh, “Dynamic permissions based android malware detection using machine learning techniques,” in *Proceedings of the 10th Innovations in Software Engineering Conference*, pp. 202–210, Jaipur, India, February 2017.

[30] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: effective and explainable detection of android malware in your pocket,” *In Ndss*, vol. 14, pp. 23–26, 2014.

[31] N. Mclaughlin, A. Doupé, G. J. Ahn, J. M. D. Rincon, and Z. Zhao, “Deep android malware detection,” in *Proceedings of the Acm on Conference on Data & Application Security & Privacy*, Richardson, TX, USA, March 2017.

[32] R. Vinayakumar, K. P. Soman, P. Poornachandran et al., “Detecting android malware using long short-term memory (lstm),” *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1277–1288, 2018.