
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2023 SCPE. Volume 24, Issues 3, pp. 521–530, DOI 10.12694/scpe.v24i3.2163

COMPUTER MALICIOUS CODE SIGNAL DETECTION BASED ON BIG DATA TECHNOLOGY

XIAOTENG LIU*

Abstract. The article addresses the challenges modelled by the inadequacy of traditional detection methods in effectively handling the substantial volume of software behavior samples, particularly in big data. A novel approach is proposed for leveraging big data technology to detect malicious computer code signals. Additionally, it seeks to attack the issues associated with machine learning-based mobile malware detection, namely the presence of a large number of features, low accuracy in detection, and imbalanced data distribution. To resolve these challenges, this paper presents a multifaceted methodology. First, it introduces a feature selection technique based on mean and variance analysis to eliminate irrelevant features hindering classification accuracy. Next, a comprehensive classification method is implemented, utilizing various feature extraction techniques such as principal component analysis (PCA), Kaehunen-Loeve transform (KLT), and independent component analysis (ICA). These techniques collectively contribute to enhancing the Precision of the detection process. Recognizing the issue of unbalanced data distribution among software samples, the study proposes a multi-level classification integration model grounded in decision trees. In response, the research focuses on enhancing accuracy and mitigating the impact of data imbalance through a combination of feature selection, extraction techniques, and a multi-level classification model. The empirical results highlight the effectiveness of the proposed methodologies, showcasing notable accuracy improvements ranging from 3.36% to 6.41% across different detection methods on the Android platform. The introduced malware detection technology, grounded in source code analysis, demonstrates a promising capacity to identify Android malware effectively.

Key words: Android malware detection, Feature extraction, Set classification algorithm, PCA, Kaehunen-Loeve transform (KLT), Independent component analysis (ICA)

1. Introduction. With the advent of the first Von Neumann computer and the 20th-century rise of the Internet to today's interconnected society, computers' significance in social life has steadily escalated. Concurrently, as computers have progressed, an unceasing influx of network security incidents has arisen. Such incidents can result in material losses for individuals and organizations and grave consequences, including personal privacy breaches and data leaks. According to the security report spanning August 2021 to January 2022 by the National Internet Emergency Response Center, a statistical chart depicting security events becomes evident. Over this half-year period, malicious program events constituted nearly a third of China's total security incidents, consistently ranking among the top two alongside vulnerability events in network security [5]. In contemporary society, the Internet bears substantial social and economic value. Yet, specific unlawful accesses enable illegal actors to hijack computers, private data, etc., yielding economic or political gains. A case in point is the WannaCry ransomware, which encrypts and extorts data from hospitals, banks, and other institutions, and the Stuxnet worm, which increased in Iran to target critical energy facilities. The former encrypts the victim host computer data and forces users into paying for data decryption. At the same time, the latter disrupts uranium plant centrifuges through a computer virus, indirectly impacting Iran's nuclear program to achieve political objectives [2].

Traditional malware detection approaches centered around feature matching and heuristic scanning fall short in adapting to the dynamic evolution of malware. Modern malicious software frequently employs tactics like shell techniques and encryption to elude conventional detection methods, necessitating more advanced technologies for effective detection. The robust capacity of machine learning to discern latent correlations between features has prompted an increasing number of experts to integrate machine learning techniques into malware identification. The malware detection approach rooted in machine learning capitalizes on conventional static and dynamic detection mechanisms to extract malware attributes. These attributes are subsequently translated

*Xinxiang vocational and technical college, Xinxiang Henan, 453000, China (XiaotengLiu5@163.com).

into formats like images and text, allowing for identification through classical machine learning classification algorithms, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other methodologies, yielding commendable outcomes. Recent years have witnessed notable strides in entity relationship extraction, primarily due to the advancement of graph convolutional networks (GCNs). Scholars have embarked on incorporating graph neural networks into pertinent domains, notably including the domain of malware detection. Notably, control flow graphs (CFGs) and function call graphs (FCGs) of binary files have emerged as pivotal starting points in this research direction within the malware detection field [10, 11].

The paper is structured into five main sections. Section 1 introduces the study's focus on computer malicious code signal detection using big data technology. Section 2 presents a comprehensive literature review to contextualize the research. In Section 3, the proposed method for detection is detailed. Section 4 encompasses the presentation and discussion of the results obtained. Finally, Section 5 provides the concluding remarks summarizing the findings and contributions of the study.

2. Literature Review. A load decomposition system is introduced that efficiently connects high-frequency data from battery-powered particles equipped with current sensors, complementing the infrastructure's low-frequency data acquired from energy meters [12]. Machine learning techniques are employed to classify executable files, including a k-nearest neighbor, support vector machine (SVM), and decision tree methods. The effectiveness of this approach was measured against dynamic detection methods that utilize dynamic system call sequences. Empirical findings indicate that this approach demonstrates enhanced detection accuracy and a reduced false positive rate compared to dynamic detection methods [3]. A malware detection approach is discussed in risk theory, incorporating feature extraction and synthesis. Using the n-gram algorithm, features are extracted from API call sequences during malware runtime and consolidated into risk and security signals. Ultimately, the deterministic dendritic cell algorithm is employed for malware detection. Experimental results highlight that, in contrast to four other detection algorithms, the proposed method displays a lower false positive rate and false negative rate [7].

Since Kaspersky discovered SMS Trojans in August 2010, Android malware has evolved rapidly. Delving into Android malware reveals that the latest malware families exhibit diverse malicious behaviors, encompassing encrypted payloads, code obfuscation, concealed commands, and manipulation of communication channels. As cutting-edge technologies advance and the black market's allure grows, the design of Android malware has become increasingly intricate and ingenious. To safeguard user information, a surge of technologies and research efforts are being channeled into malware prevention and detection [14].

Regarding feature extraction methods, two primary approaches prevail: static analysis and dynamic analysis. The dynamic detection approach involves executing Android software within controlled environments like virtual machines or sandboxes, observing software behavior and entire execution processes to discern potential malicious activities. This approach is capable of identifying anomalies in zero-day vulnerabilities. However, it might fail to induce malicious behavior in spyware adept at evasion. Additionally, dynamic detection necessitates substantial hardware resources, system allocations, and extended detection cycles.

Conversely, static detection for Android malware typically entails scrutinizing software source code or decompiled executable files to extract features and achieve malware detection without code execution. Static detection is a software analysis technique that uncovers defects or vulnerabilities by analyzing source code or executable files. In contrast to dynamic detection, static detection doesn't demand code execution, allowing it to identify potential issues and rendering it more practical rapidly. This method is widely adopted by researchers in the Android malware detection domain due to its speed, efficiency, and minimal resource requirements [15].

Many specialized big data-driven techniques for detecting malware are designed explicitly for mobile software operations. This encompasses a comprehensive exploration of Dalvik, API, and permission-based profiling of malware targeting Android systems. The investigation explores deeply into feature selection, covering frequency-averaged and variance-based algorithms. Moreover, the study extensively explores feature extraction methods, including PCA, KLT, and ICA. The research also examines a decision tree-powered multi-level classification fusion algorithm strategically devised to rapidly and accurately identify instances of mobile malware within the Android platform [18].

3. Proposed Methodology. Mobile malware detection systems rely on specialized selection algorithms rooted in mean and variance analysis, content extraction-driven malware detection algorithms, and multi-

level mixture-based malware detection algorithms. A mobile malware detection system serves the purpose of identifying malicious software on mobile devices. Given the escalating integration of mobile devices into everyday activities and the escalating risk posed by malicious software, developing such a detection system holds immense importance [9, 13]. The function of each component within the system is delineated as follows.

1) Feature extraction: It extracts the Permission feature, API feature and Dalvik feature from the collected samples.

2) Set learning: multi-level integration is adopted for classification and fusion [22, 17].

3.1. Mean-Variance-based feature selection algorithm. The Dalvik directive is in the Android software's runtime on the Dalvik virtual machine, offering information that lies deeper within the system than the API. It presents the Android software's implementation through its registry, offering the benefits of reduced instructions and simplified withdrawal, rendering it adept for effective malware detection. This section explores the efficacy of Dalvik's specialized instructions for detecting malware. During the compilation of an Android app, Java bytecode is automatically transformed into instructions tailored for the Dalvik VM, stored within the class dex. This classes.dex file is intrinsic to every Android application and is executable directly on the Dalvik VM. During runtime, the Dalvik VM accesses register operands using Dalvik instructions, encompassing null operation (nop), data operation (move), return, data definition (const), method call (invoke), data conversion, and data operation instructions. This paper's approach relies on representing the relative frequency of Dalvik instructions and employs a feature selection algorithm [1].

Disparities in the relative frequencies of Dalvik instructions hold significant implications for software vulnerability assessment. The mean relative frequency of specific Dalvik instructions within malware surpasses the normal software, indicating that malware frequently invokes specific Dalvik instructions more often than regular software. Moreover, the instruction divergence among certain malware instances surpasses that within standard software, underscoring distinct calling patterns for Dalvik instructions among different malware strains. This paper introduces Dalvik message features, involving averaging and numeric system selection.

The extent of divergence between malware and normal software regarding the relative frequency of invoking Dalvik instructions correlates with enhanced performance. Simultaneously, improved proximity between sample points of the same category signifies better relative frequency within samples of the same classification, alongside greater separation between sample points of disparate categories. The paper incorporates a linear discriminant analysis (LDA) formula to assess the classification potency of diverse Dalvik instructions, as illustrated in Equation (3.1).

$$J(A) = \frac{|\mu_b - \mu_m|^2}{S_b^2 + S_m^2} \tag{3.1}$$

where μ_b and μ_m represents the average value of the relative frequency of standard software and malicious software, respectively; S_b and S_m represents the variance of standard and malicious software's relative frequency. It can be seen from Equation (3.1) that the goal of feature selection is to maximize the $J(A)$ value, that is, to minimize the sum of variances between the two types of samples and to maximize the difference between the average values of the two types of samples. We calculate the $J(A)$ value for each Dalvik instruction. The larger $J(A)$ value is, the stronger the classification ability of feature A, and sort the features in descending order according to the size of the $J(A)$ value. Finally, the first k Dalvik instruction features is selected as a practical feature subset [21, 6].

3.2. Feature extraction-based malware detection algorithm. The malware detection algorithm centered around feature extraction employs machine learning and other techniques to identify malware through feature analysis. The fundamental concept involves initially extracting various attributes from malicious software, such as API call sequences, permission utilization, and code structure. Subsequently, these features are fed into a classifier for training, yielding a model capable of recognizing malicious software. Eventually, this model is utilized to categorize unfamiliar software. A content extraction algorithm highlights distinctions in features across diverse locations to enhance detection accuracy. Figure 3.1 illustrates the proposed novel malware detection strategy based on feature extraction.

Initially, the extraction of Dalvik instructions from the code is performed. Subsequently, employing fundamental analysis, KLT, and ICA, the original Dalvik instructions are mapped to corresponding positions,

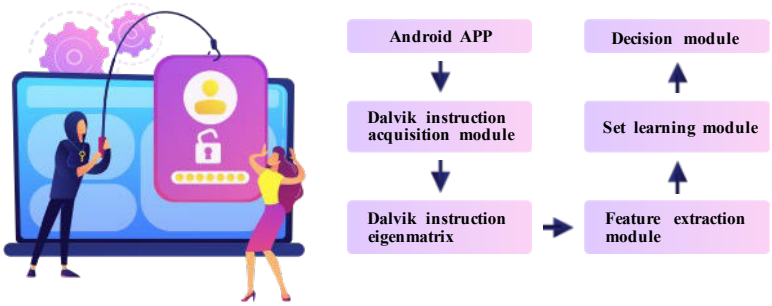

Fig. 3.1: Flow diagram of malware detection system based on feature extraction

generating three novel instructions. Lastly, a single-layer neural network is trained using a rapid machine learning algorithm as the foundational classifier. The stacking process then integrates the overall distribution base [4].

The three distinct feature extraction algorithms used to map features into distinct feature spaces are as follows:

(1) PCA: It is a linear projection technique that maps high to low dimensional features, and it is a widely used data reduction technology. Its purpose is to extract the most essential features from high-dimensional data and convert them into vectors in low-dimensional space to realize data visualization and simplify analysis. In the process of projection, the variance of data is the largest. Define N samples, $X = [X_1, X_2, \cdots, X_N]$ where; $X_i = [X_{i,1}, X_{i,2}, \cdots, X_{i,d}]^{\mathrm{T}} \in \mathrm{R}^d$; d is the number of features. The principal component analysis transforms linearly into a new sample, as follows in Equation (3.2):

$$y_i = U^{\mathrm{T}} X_i \tag{3.2}$$

where U is an orthogonal matrix. PCA initially computes the sample's covariance matrix. Subsequently, it determines the eigenvectors of this covariance matrix and transforms the original features.

(2) Kaehunen-Loeve transform: In PCA, the transformation matrix is the covariance matrix of the sample. In the Kaehunen-Loexe transformation, the transformation matrix is the inter-class dispersion matrix, which is recorded as S_w given in Equation (3.3):

$$S_w = \sum_{i=1}^{L} P_i E\left[(X - \bar{m}_i)(X - \bar{m}_i)^{\mathrm{T}}\right] \tag{3.3}$$

where L is the sample category; P_i is the probability of category i; E stands for mathematical expectation; m_i is the mean of category i. Calculate the eigenvector of S_w, and then calculate the new feature according to formula (3.2).

(3) Independent component analysis: The independent information is extracted from original features. The independent component analysis model is recorded using Equation (3.4):

$$X = As \tag{3.4}$$

where X is the original data; A is a full rank matrix, S is an independent component. The purpose of independent component analysis is to extract independent component S from X is represented in Equation (3.5):

$$\hat{s} = UX \tag{3.5}$$

where $\hat{s}$ represents the estimated value of the independent component; U is the transformation matrix and fast ICA algorithm is used to calculate U.

3.2.1. Set classifier based on extreme learning machine. Utilizing the extreme learning machine (ELM) algorithm, the single-layer neural network is employed as a foundational classifier. The Stacking technique is then employed to construct a set classifier. Noteworthy advantages of the ELM ensemble classifier encompass:

1. Swift Training Pace: The ELM algorithm's training process entails solving a linear equation system and does not require iterative optimization. Consequently, it achieves rapid training, proving especially suited for extensive datasets.
2. Elevated Classification Precision: The ELM algorithm boasts robust generalization capabilities and is adept at addressing intricate scenarios like high dimensionality, nonlinearity, and noise interference, resulting in high classification accuracy.
3. Robust Scalability: The ELM algorithm integrates with other machine learning methods, such as SVM and random forests. Its applicability extends to multi-sample, multi-feature, and multi-modal datasets.

The swift learning machine algorithm randomly assigns initial input weight vectors and biases to the neural network. Subsequently, the neural network's output weights are determined through analysis. The algorithmic sequence is as follows:

In the training stage, input weights and deviations are randomly assigned to calculate the output of hidden layer nodes, as shown in the following formula (3.6):

$$h_{ij} = g\left(w_j x_i + b_j\right) i = 1, 2, \cdots, N; j = 1, 2, \cdots, k(6) \tag{3.6}$$

where h_{ij} is the output of the j hidden layer node; $w_j = [w_{j1}, w_{j2}, \cdots, w_{jn}]^{\mathrm{T}}$ is the weight value connecting the j-th hidden node and the input data; b_j is the node deviation of the j-th hidden layer; N is the number of samples; k is the number of hidden layer nodes; g is the activation function. The output matrix of the hidden layer is marked as $H = \{h_{ij}\}$, and the weight vector connecting the hidden layer and the output layer nodes is marked as β, as follows (3.7):

$$\hat{\beta} = H^{\dagger} T \tag{3.7}$$

where $\hat{\beta}$ is the estimated value of $\beta; H^{\dagger}$ is the generalized inverse matrix of H 's Moore-Penrose; T is the classification label as given by Equation (3.8). In the test phase, for unknown samples, the hidden layer node output H is calculated first, and then its label is predicted.

$$T = \hat{\beta} H \tag{3.8}$$

where T is the prediction label of unknown samples; H is the hidden layer output of unknown samples, calculated according to Equation (3.6). This paper uses the Stacking method to build a set classifier and stacking fuses of each base classifier based on the learning method, including two layers marked as level-0 and level-1. In level-0, an extreme learning machine builds N basis classifiers. In level-1, the tags predicted in level-0 are used as input data, and the speed learning machine is also used for training [16].

3.3. Multi-level integration-based algorithm for mobile malware detection. Android software permissions and API functions are pivotal in multi-level integrated mobile malware detection. An in-depth analysis uncovers distinctions in the frequency of usage of permitted functions and API functions between normal software and malware. Hence, this article selects and compares the top 20 permissions and APIs, with the outcomes presented in Figures 3.2 and 3.3. In practical scenarios, the quantity of genuine software far exceeds that of malicious software, creating an inequality in data distribution. Generally, approaches addressing uncertain data involve resampling, large-sample size reduction, SVM, and Random Forest. Primarily, the focus centers on detecting malware, endeavoring to minimize instances of malware being wrongly identified as normal software. This involves diminishing false positives. Consequently, the prevailing approach involves customizing diverse training and combination methodologies to cater to distinct models [19].

It can be seen from Figure 3.2, the normal software and malicious software have a high frequency of applying for permissions, such as WRITE_EXTENRNAL_STORAGE and ACCESS_NETWORK_STATE, but the difference between them is small; However, the frequency of malicious software applications for

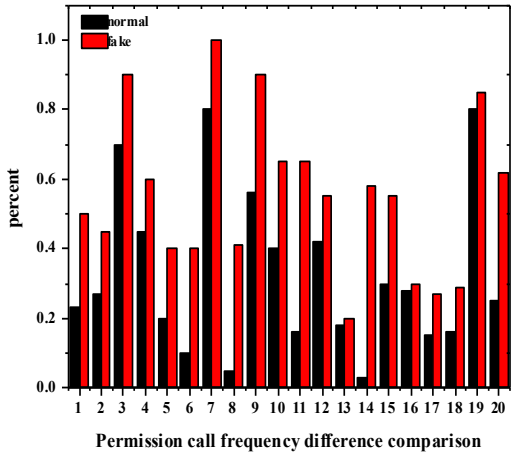

Fig. 3.2: Comparing the disparity in call frequency between normal and malicious software

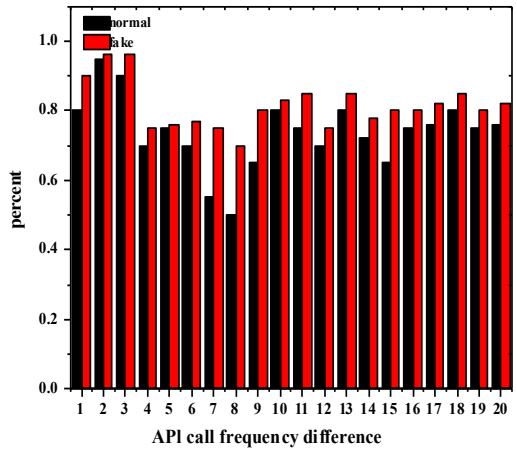

Fig. 3.3: Contrasting API call frequencies in normal and malicious software

READ_CONTACTS, SEND_SMS and other permissions is much higher than that of normal software applications. Therefore, permission is effective as a feature to distinguish normal software from malicious software.

The observation from Figure 3.3 indicates that both normal and malicious software frequently raise APIs like Android.content.Content and android.app.Activity. Despite this, the disparity between their usage frequencies remains minimal. In contrast, malware exhibits a higher frequency of invoking APIs such as android.graphics.Paint and android.View.KeyEvent compared to the frequency seen in normal software [8].

To address the issues as mentioned earlier and enhance detection accuracy, we present a novel approach known as decision tree ensembles-based detection (DTED) method. This integration technique involves a three-tier ensemble of decision trees. Among the decision tree-based integrated learning methods, the random forest algorithm is renowned. Initially, the random forest algorithm extracts a specific portion of samples and features randomly from the original data, constructing multiple decision trees, each trained on distinct sample and feature sets. During classification, test samples are input into each decision tree for classification, with the outcomes from each decision tree being collectively voted upon or subjected to weighted averaging to derive the ultimate classification result. In regression scenarios, the prediction outcomes of individual decision trees are amalgamated using an averaging approach. The framework for mobile malware detection is proposed using

Table 4.1: Composition of datasets

Dataset	Malware	Regular software	Total
D1	29102	29120	58222
D2	1262	1262	2524

the following three components:

(1) The initial set of combinations employs the complete voting approach, ensuring an equitable distribution across all decision trees. This balance mitigates ambiguous information and addresses the issue at hand.

(2) The subsequent stage of amalgamation employs a majority voting model. For models not identified in the first stage, they are combined with the majority vote of the first-stage decision trees. This integration occurs alongside the processing of different effective models from the second stage. This technology effectively reduces malware false alarms.

(3) The third layer of integration leverages a select few votes. Typically, following the two preceding detection layers, the remaining undetected samples tend to be more resilient. In such cases, where robust feature characterization is lacking, incorporating a few experienced votes can enhance accuracy to a certain extent.

Compared with classical data mining algorithms, DTED effectively addresses unbalanced data, offers high accuracy, and maintains a low false alarm rate. This success stems from the fact that the first-level integration method can effectively detect most positive samples, resulting in a balanced proportion of positive and negative samples, thus tackling the imbalance challenge. The second-level integration, combined with the first-level integration, bolsters the weight of negative samples, enhancing their detection rate and thereby diminishing the false alarm rate of negative samples. The third integration layer fortifies the outcomes from the second tier, heightening the overall detection accuracy. In summary, the DTED method excels in handling imbalanced data, and delivering high accuracy.

4. Results and Discussion.

4.1. Data sets and evaluation indicators. The D1 dataset encompasses models from Google Play, Peapod, and Android Online, totaling 29,102 models. On the other hand, the malware samples originate from the Android Malware Genome Project and Andro MalShare, amounting to a total of 29,120 samples. Malware samples for the D2 installation file dataset are sourced from the Android Malware Genome Project, while the source software exclusively comes from Google Play. Notably, dataset D1 is utilized for testing the feature selection algorithm, whereas dataset D2 serves as the testing ground for the feature extraction algorithm, as illustrated in Table 4.1.

Numerous classification systems exhibit distinct characteristics, with diverse attributes across different methods classes and performance outcomes varying based on data types. The realm of classification algorithms encompasses an array, including decision trees, naive Bayes, SVM, logistic regression, random forest, neural networks, and more. Each algorithm possesses unique merits, drawbacks, and suitable contexts, with algorithm selection contingent upon specific circumstances. The choice of classification hinges on the task's nature, the methodology for classification selection, and the approach to evaluating algorithm efficacy. Generally, Precision and F-measure stand as chosen measurement parameters. To begin, we will introduce various evaluation models. For now, let's consider our target categories as positive and negative groups [20].

1. True positive (TP): The number of events that are positive events, divided by classification into good conditions;
2. False positive (FP): the number of cases misclassified by positive cases, i.e. divided by the number of cases classified as negative but in the excellent condition;
3. False negative (FN): the number of cases classified as negative, i.e., the number of cases that were good but classified as negative by category;

Table 4.2: Classification accuracy of different feature selection algorithms

Classifier	Accuracy (%)				
	No feature selection	**Sequential Forward Selection**	**Mountain climbing algorithm**	**Relief algorithm**	**Proposed Feature Selection based on Mean and Variance**
Logistic regression	81.57	63.57	78.81	82.43	82.70
C4.5 Decision tree	87.04	77.75	86.95	87.53	87.61
Random forest	83.98	79.81	82.26	90.39	90.56

Table 4.3: Comparative performance of classifiers based on accuracy (%) and F-measure (%)

Classifier	**Accuracy (%)**	**F-measure (%)**
ELM	93.56	93.43
ELM+PCA	92.10	93.81
ELM+KLT	93.78	90.73
ELM+ICA	93.55	92.95
Stacking	95.12	93.58
Proposed FES	97.52	97.52

4. True negative (TN): The number of events correctly classified as negative events, that is, the number of negative events divided by negative events by category. The classification criteria are as follows.
   (a) Precision: Precision is a measure of accuracy, which represents the proportion of positive cases in the cases divided into positive cases, "Precision"=TP/(TP+FP);
   (b) Recall rate: recall rate is the degree of coverage, "Recall"=TP/(TP+FN);
   (c) F-measure: F-measure is the average harmonic number of recall and precision is expressed as Equation (4.1):

$$\text{F-measure} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{4.1}$$

4.2. Performance comparison of various classifiers. To evaluate the proposed FSMV classification system, this study employs regression logic, C4.5 decision trees, and random forest as classifiers, comparing them with subsequent selection algorithms (SFS), hill climbing, and Relief algorithms. Therefore, the classification outcomes using D1 data are presented in Table 4.2. The data in Table 4.2 demonstrates the enhancement in the detection capabilities of various classification systems facilitated by the specialized selection algorithm based on the mean and variance (FSMV) of Dalvik instructions. The combination of FSMV with random forest for detecting Dalvik definitions showcases a 6.41-fold improvement over the sole use of the random forest algorithm. This improvement lies below the Relief algorithm's performance and surpasses that of the C4.5 decision tree when identifying regression logic connections accurately.

To test the malware system based on the extracted content, this paper uses the D2 dataset, randomly selecting 80 files as training data and 20 files as testing data. The corresponding results are shown in Table 4.3. The table shows that the F-value of ELMPCA is higher than that of the ELM algorithm alone, and the F-value of ELMKLT and ELMICA is lower than that of ELM alone; The accuracy and F-rate of using only Stacking is higher than that of using ELM only, and the content extraction-based Stacking method is higher than other methods, indicating that it performs well.

To evaluate the malware detection method based on multi-level integration, this study employs dataset D1, randomly selecting 80% of the data for training and 20% for testing. Typically, in imbalanced data, low

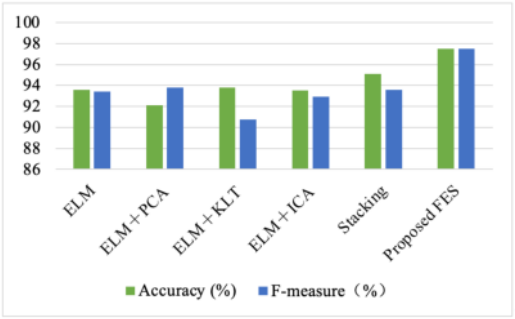

Fig. 4.1: Comparison of classification accuracy and F-measure for different classifiers and feature extraction techniques

Table 4.4: Classification accuracy and F-measure value of different algorithms

Detection algorithms	**Accuracy (%)**	**F-measure (%)**
J48 decision tree	87.37	88.24
SVM	88.13	87.20
Random forest	89.87	89.15
Proposed DTED	91.60	91.13

omission rate, and accuracy requirements, the F-measure value is widely employed as the evaluation metric. The results are displayed in Table 4.4.

The provided table presents the outcomes of classification accuracy and F-measure achieved through diverse classifiers and feature extraction techniques. The ELM classifier achieves an accuracy of 93.56% and an F-measure of 93.43%. When coupled with different feature extraction methods, ELM yields variable outcomes. "ELM+PCA" demonstrates an accuracy of 92.10% and an F-measure of 93.81%, "ELM+KLT" achieves an accuracy of 93.78% but a comparatively lower F-measure of 90.73%, and "ELM+ICA" results in an accuracy of 93.55% and an F-measure of 92.95%.

Remarkably, the "Stacking" technique achieves an accuracy of 95.12% and an F-measure of 93.58%. Nevertheless, it's the feature extraction system (FES) that boasts the highest accuracy at 97.52%, coupled with an impressive F-measure of 97.52%. These outcomes underscore the effectiveness of distinct classifiers and their combinations with varied feature extraction techniques in terms of accuracy and F-measure for malware detection. Among these, the "Proposed FES" strategy emerges as the most accurate classifier in this particular context.

The table shows that the proposed DTED method surpasses commonly utilized solutions for addressing imbalanced data, such as SVM and random forest. Notably, the DTED method exhibits higher accuracy and F-measure values than alternative approaches. This outcome highlights that the DTED method excels in handling uneven data and meeting the demands for low omission rates and high accuracy. Thus, the DTED method showcases a strong and effective detection performance.

5. Conclusion. The rapid expansion of the mobile Internet has facilitated the explosion of mobile malware, mainly targeting the Android platform due to its open nature. Consequently, evaluating the security of applications released on the internet and app stores and discriminating malicious software remains a pressing research concern. This study critically examines the current challenges of mobile terminal malware detection technology. It introduces a novel approach by scrutinizing software attributes from a source code perspective while investigating pivotal facets of Android malware detection, encompassing Dalvik instructions, permissions, and APIs. Essential techniques such as feature selection and extraction algorithms, alongside multi-level

integrated classification methods, were meticulously explored. Empirical findings validate the proposed source-based malware detection technology, confirming its efficiency in accurately identifying Android malware. This research contributes substantially to the ongoing discourse by presenting a multi-layered solution that addresses the growing threat of mobile malware. Connecting source-based detection and comprehensive methodologies offers a robust safeguard for the Android platform's security and integrity in the evolving mobile Internet.

REFERENCES

[1] M. M. Alani and A. I. Awad, *Paired: An explainable lightweight android malware detection system*, IEEE Access, 10 (2022), pp. 73214–73228.

[2] S. M. Bellovin, S. Landau, and H. S. Lin, *Limiting the undesired impact of cyber weapons: technical requirements and policy implications*, Journal of Cybersecurity, 3 (2017), pp. 59–68.

[3] Y. Ding, X. Yuan, D. Zhou, L. Dong, and Z. An, *Feature representation and selection in malicious code detection methods based on static system calls*, Computers & Security, 30 (2011), pp. 514–524.

[4] E.-S. M. El-Kenawy, S. Mirjalili, F. Alassery, Y.-D. Zhang, M. M. Eid, S. Y. El-Mashad, B. A. Aloyaydi, A. Ibrahim, and A. A. Abdelhamid, *Novel meta-heuristic algorithm for feature selection, unconstrained functions and engineering problems*, IEEE Access, 10 (2022), pp. 40536–40555.

[5] D. A. Fernandes, L. F. Soares, J. V. Gomes, M. M. Freire, and P. R. Inácio, *Security issues in cloud environments: a survey*, International Journal of Information Security, 13 (2014), pp. 113–170.

[6] A. Hashemi, M. B. Dowlatshahi, and H. Nezamabadi-pour, *Ensemble of feature selection algorithms: a multi-criteria decision-making approach*, International Journal of Machine Learning and Cybernetics, 13 (2022), pp. 49–69.

[7] C. Huang, J. Chen, S. Gong, Q. Luo, and Q. Zhu, *Feature representation and selection in malicious code detection methods based on static system calls*, Journal of Central South University(Science and Technology), 45 (2014), pp. 3055–3060.

[8] U. M. Khaire and R. Dhanalakshmi, *Stability of feature selection algorithm: A review*, Journal of King Saud University-Computer and Information Sciences, 34 (2022), pp. 1060–1073.

[9] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, *Mapas: a practical deep learning-based android malware detection system*, International Journal of Information Security, 21 (2022), pp. 725–738.

[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, 14 (2018), pp. 773–788.

[11] V. Kouliaridis and G. Kambourakis, *A comprehensive survey on machine learning techniques for android malware detection*, Information, 12 (2021), p. 185.

[12] M. Kumar, *Scalable malware detection system using big data and distributed machine learning approach*, Soft Computing, 26 (2022), pp. 3987–4003.

[13] S. R. T. Mat, M. F. Ab Razak, M. N. M. Kahar, J. M. Arif, and A. Firdaus, *A bayesian probability model for android malware detection*, ICT Express, 8 (2022), pp. 424–431.

[14] M. Mijwil, I. E. Salem, and M. M. Ismaeel, *The significance of machine learning and deep learning techniques in cybersecurity: A comprehensive review*, Iraqi Journal For Computer Science and Mathematics, 4 (2023), pp. 87–101.

[15] A. Mitra, B. Bera, A. K. Das, S. S. Jamal, and I. You, *Impact on blockchain-based ai/ml-enabled big data analytics for cognitive internet of things environment*, Computer Communications, 197 (2023), pp. 173–185.

[16] W. Qian, Y. Xiong, J. Yang, and W. Shu, *Feature selection for label distribution learning via feature similarity and label correlation*, Information Sciences, 582 (2022), pp. 38–59.

[17] J. Qiu, Q.-L. Han, W. Luo, L. Pan, S. Nepal, J. Zhang, and Y. Xiang, *Cyber code intelligence for android malware detection*, IEEE Transactions on Cybernetics, 53 (2022), pp. 617–627.

[18] S. Sahoo, *Big data analytics in manufacturing: a bibliometric analysis of research in the field of business management*, International Journal of Production Research, 60 (2022), pp. 6793–6821.

[19] R. Sudharsan and E. Ganesh, *A swish rnn based customer churn prediction for the telecom industry with a novel feature selection strategy*, Connection Science, 34 (2022), pp. 1855–1876.

[20] A. Thakkar and R. Lohiya, *A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions*, Artificial Intelligence Review, 55 (2022), pp. 453–563.

[21] Y. Wu, J. Shi, P. Wang, D. Zeng, and C. Sun, *Deepcatra: Learning flow-and graph-based behaviours for android malware detection*, IET Information Security, 17 (2023), pp. 118–130.

[22] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, *A two-stage deep learning framework for image-based android malware detection and variant classification*, Computational Intelligence, 38 (2022), pp. 1748–1771.

Edited by: C. Venkatesan
Special Issue: Next Generation Pervasive Reconfigurable Computing for High Performance Real Time Appls
Received: Mar 20, 2023
Accepted: Sep 1, 2023