JOWMDroid: Android Malware Detection Based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters

Lingru Cai , Yao Li , Zhi Xiong

PII: S0167-4048(20)30359-X
DOI: https://doi.org/10.1016/j.cose.2020.102086
Reference: COSE 102086

To appear in: *Computers & Security*

Received date: 14 June 2020
Revised date: 8 September 2020
Accepted date: 13 October 2020

Please cite this article as: Lingru Cai , Yao Li , Zhi Xiong , JOWMDroid: Android Malware Detection Based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters, *Computers & Security* (2020), doi: https://doi.org/10.1016/j.cose.2020.102086

JOWMDroid: Android Malware Detection Based on Feature Weighting with Joint Optimization of Weight-Mapping and Classifier Parameters

Lingru Cai[1,2], Yao Li[1], and Zhi Xiong[1,2,*]

[1]Department of Computer Science and Technology, Shantou University, Shantou 515063, China

[2]Key Laboratory of Intelligent Manufacturing Technology (Shantou University), Ministry of Education, Shantou University, Shantou 515063, China

*Correspondence: zxiong@stu.edu.cn

Abstract: Android malware detection is an important problem that must be urgently studied and solved. Machine learning-based methods first extract features from applications and then build a classifier using machine learning algorithms to distinguish malicious and benign applications. In most of the existing work, the difference in feature importance has been ignored, or the calculation of feature weights is irrelevant to the classification model. To address these issues, this paper proposes a novel Android malware detection scheme based on feature weighting with the joint optimization of weight-mapping and classifier parameters, called JOWMDroid. First, features of eight categories are extracted from the Android application package and then a certain number of the most important features are selected using information gain for malware detection. Next, an initial weight is calculated for each selected feature via three machine learning models and then five weight-mapping functions are designed to map the initial weights to the final weights. Finally, the parameters of the weight-mapping function and classifier are jointly optimized by the differential evolution algorithm. The experimental results reveal that the proposed method outperforms four state-of-the-art feature weighting methods and makes the weight-aware classifiers more competitive.

Keywords: Android, malware detection, feature weighting, mapping function, joint optimization

1. Introduction

Since Google released its first Android operating system in 2007, Android has gained immense popularity and has become the most popular mobile operating system in the world. According to Statcounter [1], in the third quarter of 2019, the Android mobile phone had a market share of approximately 76.67% in sales. In particular, Android users increased from 1.4 billion in 2015 to 2.5 billion in 2019, an increase of 178.57% [2]. Meanwhile, the growth of Android malware (namely, malicious software, including viruses, spyware, and other unwanted software that gets installed on your system without your consent) poses a substantial threat to users, and this situation has been serious in recent years because Android malware has become more infectious and disseminated [3]. In “TenSec 2019”, Tencent Keen Security Lab released “Android Application Security White Paper 2018.” According to the white paper, Android applications found more than 3.2 million new malicious samples, averaging more than 11,000 per day [4]. According to statistics, Android malware accounts for 17% of the total applications [5]. Therefore, Android malware detection is a key problem that must be urgently studied and solved.

Machine learning (ML) has been widely used in Android malware detection [6]. The ML-based method primarily involves two steps. The first step is to extract features from applications, and the second step is to build a classifier to distinguish malicious applications from benign applications. According to the manner of feature extraction, detection methods can be divided into two categories: static (e.g., [7]–[13]) and dynamic (e.g., [14]–[16]) analyses. Dynamic analysis must run applications on a device or simulator and monitor the runtime information of the applications, such as the network traffic and system calls. This method can achieve high accuracy in identifying malicious activities; however, the runtime monitoring is very costly and cannot be deployed on mobile devices [6]. Static analysis obtains static feature information through decompiling the installation packages of the applications (i.e., the APK files), such as permissions and application programming interface (API) calls. This method has lower overhead and higher code coverage, and it can be directly applied to mobile devices.

Almost all static analysis methods extract binary (namely Boolean) features from the APK files, such as whether a permission is used [7]–[9], [17], [18], whether a function is called [18], [19], whether a network address exists [9], [20], and whether a key subgraph exists in the function call graph [21]. Some dynamic analysis methods also use binary features, such as whether a system call is triggered [6], [21], and whether an API is called [6]. In most of these studies, the binary features are directly fed into the classifier to train the model in the training phase and to distinguish malicious applications from benign applications in the detection phase. However, the importance of features is different. For example, SEND_SMS and CALL_PHONE are dangerous permissions, while FLASHLIGHT and INSTALL_SHORTCUT are normal permissions. Therefore, the former two are more important than the latter two for malware detection. Similarly, android.telephony.TelephonyManager.listen() and android.app.ActivityManager.getRunningTasks() are sensitive APIs that can violate users' privacy, while android.media.effect() and android.hardware() are normal APIs. Therefore, it is not rational to treat all features similarly. In recent years, feature weighting has been proposed by related work [7], [8], [22], [23]. In these studies, how to measure the importance of features and how to assign an appropriate weight to each feature is critical. If the weights are not properly set, the detection accuracy may even be worse. However, these studies estimate the importance of each feature based on the statistical analysis of the original binary feature matrix. Their feature weighting is irrelevant to ML algorithms. Therefore, the accuracy of the classifier for malware detection may not be satisfactory.

To address the above issues, this paper proposes a novel Android malware detection scheme based on feature weighting with joint optimization of weight-mapping and classifier parameters, JOWMDroid. Accordingly, the proposed feature weighting method is called JOWM. The scheme is an ML-based static analysis method whose main workflow is presented in Figure 1. In the training phase, binary features of eight categories are first extracted from the APK files and then a certain number of the most important features are selected using information gain (IG) for malware detection. Next, an initial weight is calculated for each selected feature. Then, it is mapped to a final weight using a weight-mapping function. Finally, the parameters of the weight-mapping function and classifier are jointly optimized using the differential evolution (DE) algorithm. In the detection phase, the mapped weights and classifier with optimized parameters are used to identify malicious applications.

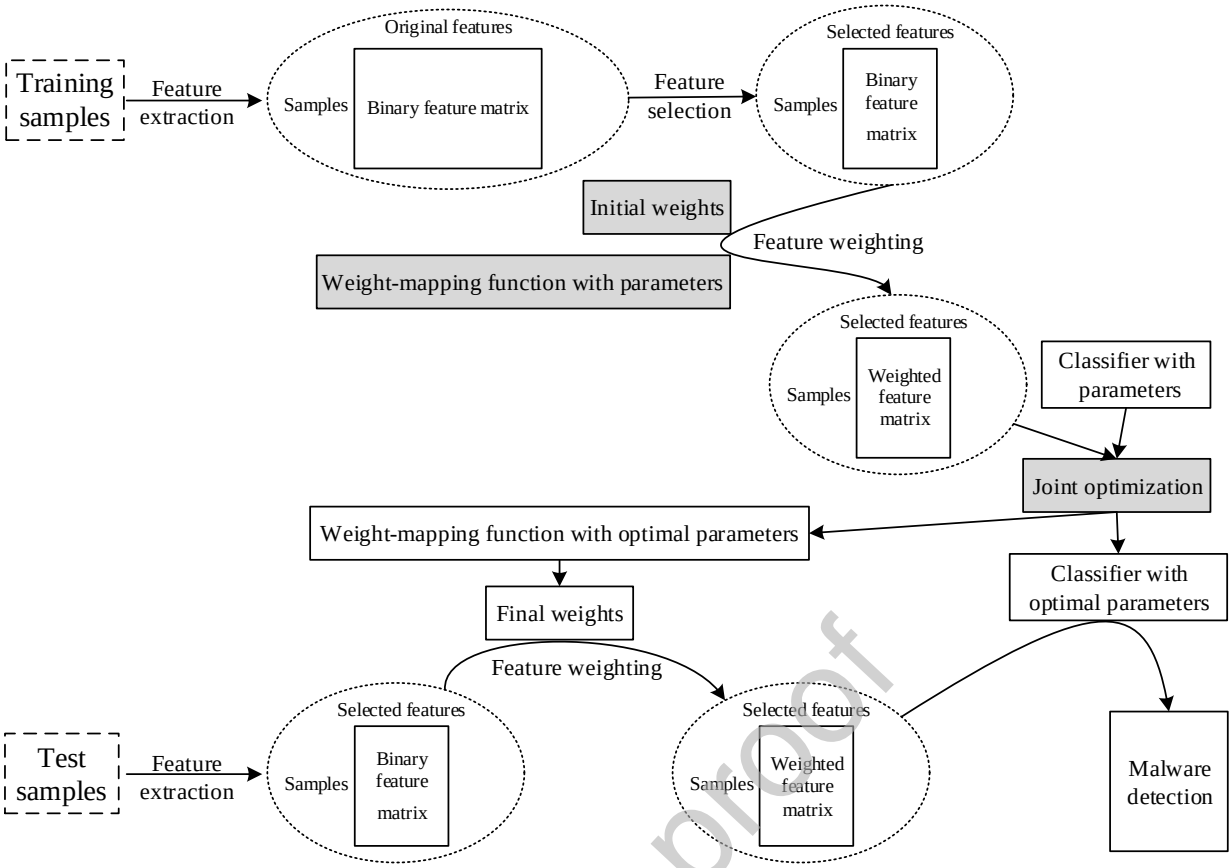


Figure 1. Main workflow of JOWMDroid

The main contributions of this paper are the following:

- This paper proposes a novel feature weighting method (i.e., JOWM). It consists of three links. The first link is to calculate an initial weight for each feature, and the second link is to map the initial weights to the final weights via a weight-mapping function. The third link is to optimize the parameters of the weight-mapping function and classifier jointly.
- The importance measures of features supplied by the three classification models are used to calculate the initial weight of each feature. That is, the proposed feature weighting is relevant to ML algorithms, which improves the accuracy of the subsequent classifier for malware detection.
- Five parameterized weight-mapping functions, namely the power, exponential, logarithmic, hyperbolic, and S-shaped curve functions, are designed to map the initial weights to the final weights. Moreover, the DE algorithm is used to jointly optimize the parameters of the weight-mapping function and classifier, which causes the classifier to achieve higher accuracy.
- This paper presents the detailed evaluation results. These results demonstrate the effectiveness of JOWM from multiple perspectives.

The rest of this paper is organized as follows. The related work is introduced in Section 2. Section 3 describes the feature extraction and selection methods. Section 4 proposes the novel feature weighting method (i.e., JOWM). The evaluation and analysis are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Related Work

Android malware detection technologies can be primarily divided into two categories: static and dynamic analyses. Dynamic analysis is based on the behavior of applications at runtime and must run the applications on a mobile device or simulator using an automated test input generation tool (such as Monkey [24]) and then monitor the running of the applications, for example, monitoring network traffic [25], monitoring system calls [26], [27], and

monitoring API calls and permission use [24]. Dynamic analysis can achieve high accuracy in identifying malicious activities, but it is very costly [6], [28]. Moreover, some applications do not allow themselves to run on simulators. Although a tool like Monkey is generally helpful in performing stress tests, it is not a sufficient approach to executing all the components (or code) of an application for malware detection [29].

Static analysis extracts static features by decompiling APK files, for example, extracting permissions [8], [17], [30], [31], extracting APIs [32], using APIs and permissions [19], and using five views such as APIs, permissions, and instructions [33]. Reference [34], [35], and [36] used decompilation techniques to obtain function call graphs (FCG) and then analyze the FCGs to obtain features. Reference [37] used control-flow graph and data-flow graph as features. Reference [38] converted an APK file into a grayscale image and then extracted its image features. However, analyzing graphs or images to extract features is a time-consuming job. The proposed scheme in this paper uses static analysis and extracts features of eight categories.

In recent years, many researchers have focused on feature weighting to improve the accuracy of Android malware detection. In information retrieval, term frequency-inverse document frequency (TF-IDF) is a numerical statistic intended to reflect how important a word is to a document in a collection or corpus. Reference [8] and [27] regarded a feature as a word and an application as a document and used TF-IDF to calculate the importance of each feature. The relevance frequency (ReF) weighting method was proposed in [7]. For a feature, its weight is calculated by the equation $log_2(1 + (a/c))$, where a is the number of applications in the malicious category that contain the feature, and c is the number of applications in the benign category that contain the feature. Entropy represents the degree of disorder or uncertainty in a system. Then, the presence of a feature in only one category results in minimum entropy, indicating that the feature might be a good discriminant feature for classification. The entropy is maximum, if the feature appears in all the categories with the same frequency [23]. Based on this idea, entropy-based category coverage difference (ECCD) was proposed to calculate a score for each feature in [23]. In [22], an effective maliciousness score of permission (EMSP) was proposed to calculate a weight for each permission. It calculates the occurrence probabilities of each permission in the benign and malware categories and then uses their subtraction to represent the maliciousness weight of the permission. However, in these methods, feature weighting is irrelevant to the classification models and ML algorithms; therefore, the accuracy of the subsequent classifier may be not satisfactory.

3. Feature Extraction and Selection

JOWMDroid uses static analysis and extracts features of eight categories. This section describes the feature extraction and selection methods.

3.1 Feature Extraction

Android installs various applications via APK files. Androguard [39] is an Android reverse engineering tool that can disassemble APK files. Figure 2 illustrates the feature extraction method. By decompressing the APK file, two key files are obtained: Manifest.xml and Class.dex. The Manifest.xml file contains the package name, permissions, version number, component information, and so on. In the scheme, the following four categories of features were extracted:

- **Feature:** A feature represents the application requirements for system resources, including hardware and software resources.
- **Requested permission:** If an application needs to use external resources or information, it must request the appropriate permissions.
- **App component:** Application components are the essential building blocks of an application, and they contain four components: activities, services, providers, and receivers.
- **Intent filter:** The intent filter specifies the type of intents that the components would like to receive.

Next, Androguard was used to convert the Class.dex file into jar files and then the following four categories of features were extracted from them:

- **Restricted API call:** Some API calls are protected by system permissions.
- **Used permission:** The access to some restricted API calls need the application to obtain some permissions.
- **Sensitive API call:** Some API calls allow access to sensitive data and resources on smartphones.
- **Sensitive shell command:** Some commands can gain root privilege and perform dangerous operations.

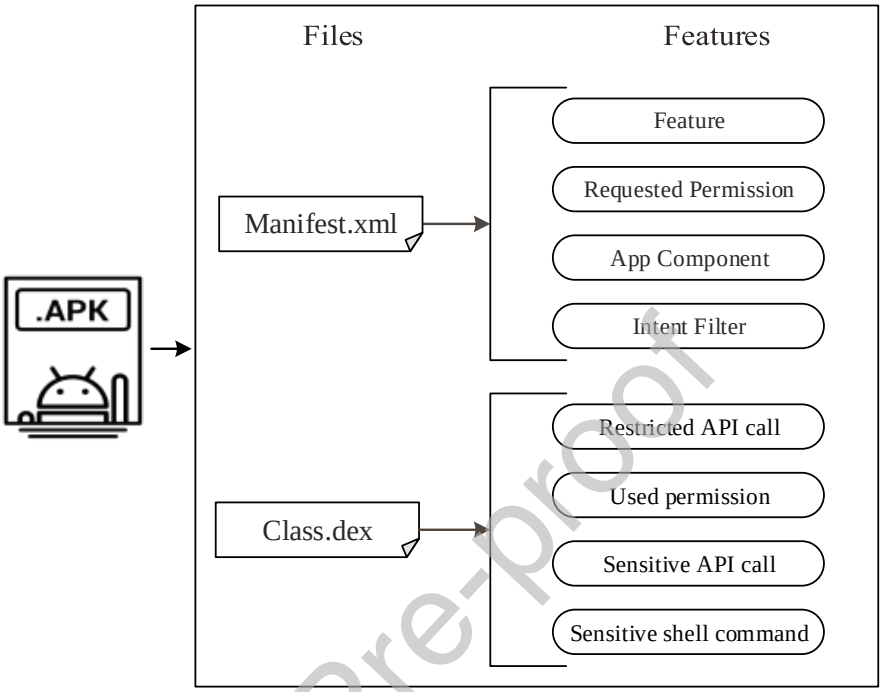


Figure 2. Feature extraction

Each feature is expressed as a string. Each feature string is divided into two parts using "::", and the first part denotes the category of the feature. For a dataset, the union of the features of all applications is called a feature set. Each application corresponds to a binary feature vector, whose length is equal to the size of the feature set. Figure 3 displays a diagram of the feature vector of an application. If a feature exists in the application, the corresponding position of the feature is marked as 1. Otherwise, it is marked as 0.

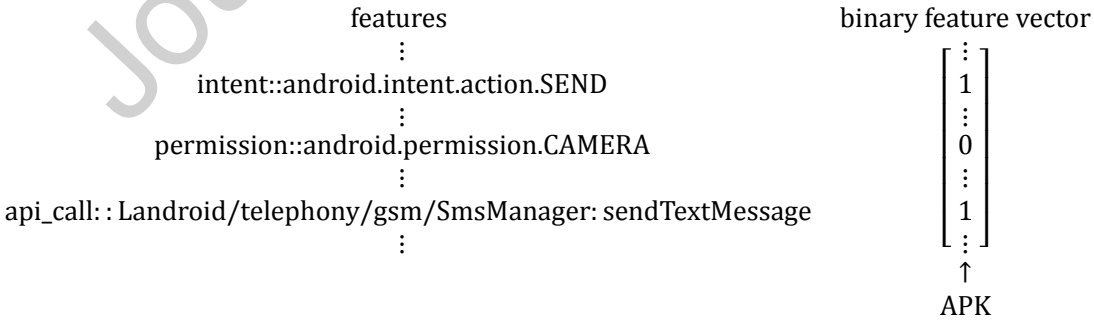


Figure 3. Binary feature vector

3.2 Feature Selection

A real dataset contains many applications, and one can extract numerous features from the APK file of each application; thus, the number of features in the original feature set is enormous. If all the original features are used to detect malware, it leads to considerable computational overhead and suffers from "the curse of dimensionality." Meanwhile, many features may be not helpful or may be less relevant to malware detection. Therefore, a subset of the most relevant features needs to be selected for malware detection. In the scheme, IG (also known as mutual

information) was used to perform feature selection.

Moreover, IG is an entropy-based feature evaluation method. As IG is used in feature selection, it is defined as the amount of information provided by the features for the application category (malicious or benign). The values of IG vary from 0 (no information) to 1 (maximum information). The features that contribute more information have a higher IG value and can be selected, whereas those that do not add much information have a lower IG value and can be removed [40]. The symbol $C = \{\text{malicious}, \text{benign}\}$ was used to denote the categories of applications and the symbol F_i was used to denote the *i*th feature; thus, the IG value of F_i was calculated as follows:

$$\text{IG}(F_i, C) = \sum_{f\in\{0,1\}}\sum_{c\in\{malicious,benign\}}\left(P(F_i = f) * P(C = c|F_i = f) * \log_2\left(\frac{P(C=c|F_i=f)}{P(C=c)}\right)\right), \quad (1)$$

where P(A) denotes the probability of event A and P(B|A) denotes conditional probability, namely, P(B|A) = P(A and B) / P(A). If the denominator is 0 in the calculation, it will be replaced by a very small number.

4. Feature Weighting

Suppose m features exist in the selected feature set: f_i, i = {1, 2, …, m}. Because the importance of features is different, a weight (w_i) was assigned to each feature (f_i) according to its importance. Correspondingly, the original binary feature vector of an application becomes a weighted feature vector. Figure 4 shows a schematic of feature weighting, i.e., each feature was multiplied by its corresponding weight. The proposed feature weighting method, JOWM, consists of three links: (i) calculating an initial weight for each feature, (ii) mapping the initial weights to final weights via a weight-mapping function, and (iii) optimizing the parameters of the weight-mapping function. Below they are described respectively.

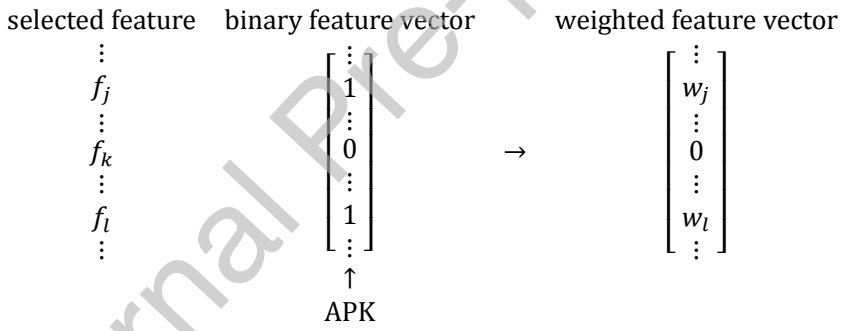


Figure 4. Weighted feature vector

4.1 Initial Weights

The IG value of each feature was calculated and used to select features in Section 3.2. However, the IG value of each feature represents the relevance between the feature and labels (namely, the categories of malicious or benign), and it may not be an appropriate weight for each feature. More importantly, the IG values have no relevance to any classification models or ML algorithms. Thus, using them as feature weights may be not helpful in improving the accuracy of the final classifier for malware detection. Therefore, classification models were used to calculate the initial weight of each feature.

After being trained, some classification models provide the importance metric of each feature, which can be used for feature selection and weight assignment. For example, in the scikit-learn (i.e., sklearn) library [41], the support vector machine (SVM) model supplies the "coef_" metric and the decision tree model supplies the "feature_importances_" metric. However, such measures supplied by a model are based on the model itself but may not suit other models well. Therefore, JOWM uses three different models and synthesizes the provided measures to calculate the initial weight of each feature. Specifically, SVM, random forest (RF), and logistic regression (LR) were used to train the data in the training set, and each model provides an importance metric for each feature f_i respectively, denoted as w_i^{SVM}, w_i^{RF} and w_i^{LR}. Considering that the scales of the metrics given by the three models

may be different and these metrics may have negative values, the metrics were normalized to a range of 0 to 1 to avoid having only one model playing a dominant role:

$$\bar{w}_i^{SVM} = \frac{w_i^{SVM} - \min_{1\le i\le m} w_i^{SVM}}{\max_{1\le i\le m} w_i^{SVM} - \min_{1\le i\le m} w_i^{SVM}} \quad (2)$$

$$\bar{w}_i^{RF} = \frac{w_i^{RF} - \min_{1\le i\le m} w_i^{RF}}{\max_{1\le i\le m} w_i^{RF} - \min_{1\le i\le m} w_i^{RF}} \quad (3)$$

$$\bar{w}_i^{LR} = \frac{w_i^{LR} - \min_{1\le i\le m} w_i^{LR}}{\max_{1\le i\le m} w_i^{LR} - \min_{1\le i\le m} w_i^{LR}} \quad (4)$$

If the denominator in Equation (2) is 0, then $\bar{w}_i^{SVM} = 0.5$. Equations (3) and (4) were treated in the same way. Finally, $\bar{w}_i^{SVM}$, $\bar{w}_i^{RF}$, and $\bar{w}_i^{LR}$ were averaged to obtain the initial weight of feature f_i:

$$w_i^{Init} = \frac{\bar{w}_i^{SVM} + \bar{w}_i^{RF} + \bar{w}_i^{LR}}{3}. \quad (5)$$

In theory, the initial weights can also be used to select features. However, the number of features in the original feature set is enormous, and it is very time-consuming to train the three classification models with all of the original features.

4.2 Weight-Mapping Function

Although the initial weights of features were calculated by three classification models, they may not be the most appropriate weights for the final classifier for malware detection. Therefore, JOWM maps the initial weights to a group of better weights via a parameterized weight-mapping function and relies on the final classifier for malware detection to choose the best parameters for the weight-mapping function.

The authors thought that the initial weights express the ranking of the importance of features well, so the basic requirement of weight-mapping is that a higher initial weight results in a greater mapped weight (i.e., weight-mapping does not change the ranking of the importance of features). Therefore, the weight-mapping function must be a monotonically increasing function. As shown in Table 1, five mapping functions were considered in this paper: the power, exponential, logarithmic, hyperbolic, and S-shaped curve functions. The expressions of the original mapping functions are listed in the second column. In the weight-mapping, the input is the initial weight (w_i^{Init}) ranging from 0 to 1, and the output is the mapped weight (w_i). It was limited to the range of 0 to $\gamma(>0)$, where γ is a parameter representing the maximum value of the mapped weights. Therefore, the outputs in the second column were translated and scaled, forming the final weight-mapping functions in the third column:

$$w_i = f(w_i^{Init}),\ w_i^{Init} \in [0,1],\ w_i \in [0,\gamma]. \quad (6)$$

Table 1. Five weight-mapping functions

Mapping function		Weight-mapping function	
Name	Expression	Expression	Parameters
Power Function (PF)	$y = x^{\alpha},\ \alpha \ge 0$	$f(x) = \gamma x^{\alpha}$	$\alpha \ge 0,\ \gamma > 0$
Exponential Function (EF)	$y = \alpha^{\beta x},\ \alpha > 1,\ \beta > 0$	$f(x) = \gamma \frac{(\alpha^{\beta x} - 1)}{\alpha^{\beta} - 1}$	$\alpha > 1,\ \beta > 0,\ \gamma > 0$
Logarithmic Function (LF)	$y = ln(1 + \alpha x),\ \alpha > 0$	$f(x) = \gamma \frac{ln(1+\alpha x)}{ln(1+\alpha)}$	$\alpha > 0,\ \gamma > 0$
Hyperbolic Function (HF)	$y = \frac{\alpha x}{1+\beta x},\ \alpha, \beta > 0$	$f(x) = \gamma \left(\frac{\alpha x}{1+\beta x}\right) \Big/ \left(\frac{\alpha}{1+\beta}\right)$	$\alpha > 0,\ \beta > 0,\ \gamma > 0$

S-shaped Curve Function (SF)	$y = \frac{1}{1+\alpha e^{-x}}\ ,\ \alpha > 0$	$f(x) = \gamma\left(\frac{1}{1+\alpha e^{-x}} - \frac{1}{1+\alpha}\right)\Big/\left(\frac{1}{1+ae^{-1}} - \frac{1}{1+a}\right)$	$\alpha > 0,\ \gamma > 0$

4.3 Joint Optimization of Weight-Mapping and Classifier Parameters

The final classifier for malware detection was relied on to optimize the parameters α, β (if existing), and γ in the weighting-mapping function. For the convenience of description, all weighting-mapping functions in Table 1 were deemed to have the three parameters: α, β, and γ, which are all real numbers. If β does not exist, it is ignored. However, the final classifier may also have some key parameters that must be optimized. For simplicity, JOWM optimizes at most one numerical parameter, denoted as δ, of the classifier. Similarly, δ is ignored if the final classifier has no key numerical parameters. Thus, four parameters must be optimized: α, β, γ, and δ.

The DE algorithm is a stochastic search and global optimization algorithm based on population evolution. It uses real number encoding and includes three basic operators: mutation, crossover, and selection. JOWM uses the DE algorithm to determine the optimal values of $s = (\alpha, \beta, \gamma, \delta)$. If δ is an integer, a rounding operator is added for it. The detailed algorithm is provided in Algorithm 1, where s is deemed to be an individual. Figure 5 illustrates the computational method of the fitness value of an individual.

Algorithm 1. The DE algorithm for finding optimal parameters

1: Parameter settings: population size *NP*, max generation *G*, scaling factor *F*, and crossover rate *CR*;

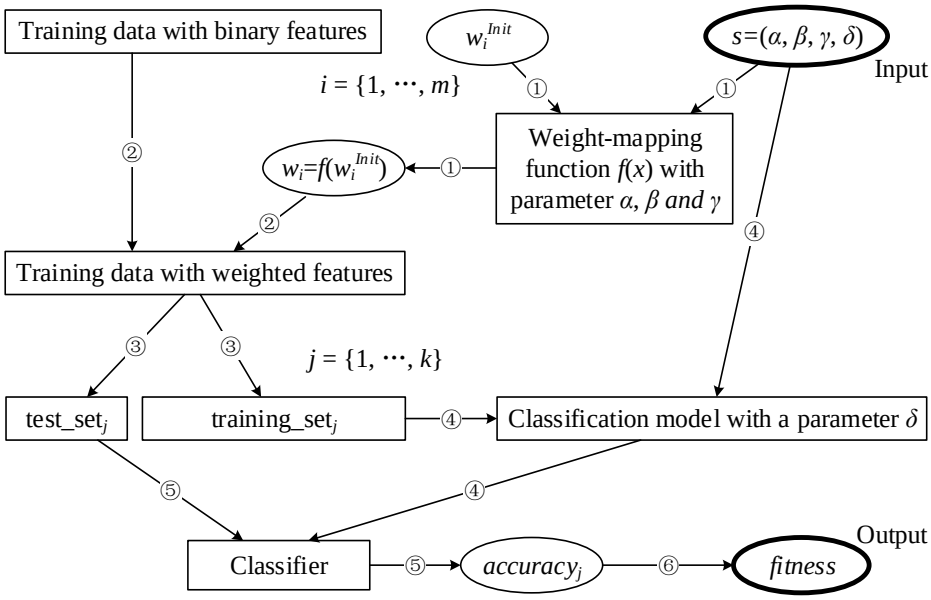
2: Randomly generate individuals $s_i, i = 1, \dots, NP$, in the initial population and current generation $g = 0$;

3: Mutation: the difference vector of two individuals within the population is scaled and then added to a different individual to produce a variation individual: $v_i = s_{r_1} + F \times (s_{r_2} - s_{r_3}),\ i \neq r_1 \neq r_2 \neq r_3$;

4: Crossover: the variation and original individuals cross each other to produce a test individual: $u_{i,j} = \begin{cases} v_{i,j}\ , & rand(0,1) \leq CR \\ s_{i,j}\ , & others \end{cases}$, where j denotes the component of the individual, and u_i is guaranteed to contain at least one component of the variation individual v_i;

5: Selection: the original individual w_i and test individual u_i are evaluated using a fitness function, and the one with a larger fitness value is the new individual w_i entering the population of the next generation, and the current generation $g = g+1$ is updated;

6: Determine whether g is equal to G, and if so, stop the iteration and output the optimal individual in the population; otherwise, repeat Steps 3 through 5.



Steps:
① Calculate the weight of each feature using the weight-mapping function
② Replace binary features with the weighted features
③ Use *k*-fold cross-validation and split the training data into training set and test set
④ Train the classification model
⑤ Test the classifier with the test set and calculate the accuracy
⑥ Average the accuracies of the *k* folds to get the fitness value of the individual

Figure 5. Computational method of the fitness value of an individual

5. Evaluation

This section presents the evaluation of the work, which includes four aspects: (i) the performance of the initial weights, (ii) the performance improvement of the weight-mapping and joint parameter optimization, (iii) the efficiency of the joint optimization, and (iv) a performance comparison with weight-unaware classifiers. Before performing these evaluations, this section first introduces the dataset and metrics and then lists the feature weighting methods and classifiers involved in the evaluation.

5.1 Dataset and Metrics

The datasets used for evaluation come from three sources. First, Drebin [28] contains 5,560 malicious applications from 179 different malware families (collected from August 2010 to October 2012) and 123,453 benign applications. Second, AMD [42] contains 24,553 malicious samples, categorized in 135 varieties among 71 malware families collected from 2010 to 2016. Third, two popular application markets were also used: Google Play [43] and APKPure.com [44]. The authors downloaded about 4,000 popular benign applications from the Google Play and AP-KPure.com that include different categories, such as social, news, entertainment, finance, education, games, sports, music, shopping, banking, and weather. From these applications, 3,000 malicious applications and 3,000 benign applications were randomly selected to perform the evaluation. From these 6,000 applications, 45,448 features were extracted. After feature selection using IG, the number of features was reduced to 643. Further, 80% of the applications were randomly selected as the training set, and the remainder was used as the testing set. In the joint optimization of weight-mapping and classifier parameters, five-fold cross-validation was used over the training set to calculate the fitness value of each individual (see Section 4.3).

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

- True positive (TP): the number of positive samples that are correctly predicted as positive.

- False negative (FN): the number of positive samples that are incorrectly predicted as negative.
- False positive (FP): the number of negative samples that are incorrectly predicted as positive.
- True negative (TN): the number of negative samples that are correctly predicted as negative.

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

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

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

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

$$fscore = \frac{2}{\frac{1}{precision}+\frac{1}{recall}} = \frac{2*precision*recall}{precision+recall}. \tag{10}$$

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

All experiments were run on a personal computer with an Intel (R) Core (TM) i7-4790 @ 3.6 GHz CPU with 8 GB of memory, and a Windows 7 64-bit operating system. The software used for evaluation includes Python 3.6.1 and sklearn 0.21.3.

5.2 Feature Weighting Methods and Classifiers in the Evaluation

In the evaluation, the proposed feature weighting method was compared against four state-of-the-art weighting methods: TF-IDF, ReF, ECCD, and EMSP, which were introduced in Section 2. As mentioned in the first paragraph of Section 4, the proposed feature weighting method, JOWM, includes three links: calculation of the initial weights, weight-mapping using a weight-mapping function, and parameter optimization of the weight-mapping function. To better demonstrate the effectiveness of each link, two feature weighting methods were derived from JOWM: (i) JOWM-IW, which only uses the initial weights but does not use the weight-mapping function, and (ii) JOWM-IO, which does not optimize the parameters of the weight-mapping function and classifier jointly but optimizes the parameters of the weight-mapping function in the joint optimization algorithm (i.e., independent optimization). Moreover, NW denotes that the no feature weighting method is used (i.e., it uses the original binary features).

According to the work principle, classifiers are divided into two categories: weight-aware and weight-unaware. For a weight-unaware classifier, feature weighting does not affect its performance. In common classifiers, the k-nearest neighbor (KNN), SVM, LR, and multilayer perceptron (MLP) methods are weight-aware, whereas Gaussian naive Bayes (NB), multinomial NB, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), and RF are weight-unaware. Therefore, KNN, SVM, LR, and MLP are used to perform the first three evaluations (Sections 5.3 to 5.5) and they are compared with Gaussian NB, multinomial NB, LDA, QDA, and RF in the last evaluation (Section 5.6). In the four weight-aware classifiers, except KNN (*k* is usually set to 1), the other three classifiers all have some key parameters. The key parameters optimized in the evaluation are explained in Table 2.

Table 2. Key parameters of classifiers

Classifier	Key parameter	Default value	Explanation
SVM	*C*: Penalty parameter of the error term	1.0	It trades off misclassification of training examples against the simplicity of the decision surface. A low *C* makes the decision surface smooth, whereas a high *C* aims at classifying all training examples correctly.
LR	*C*: Inverse of the regularization strength	1.0	Regularization is applying a penalty to increasing the magnitude of parameter values to reduce overfitting. Smaller values specify stronger regularization.
MLP	learning_rate_init: Initial learning rate	0.001	It controls the step size in updating the weights.

5.3 Performance of the Initial Weights

This subsection evaluates the performance of the initial weights but does not involve weight-mapping. Therefore, JOWM-IW was compared against TF-IDF, ReF, ECCD, EMSP, and NW. For classifier parameters, the default values were used for the moment. Tables 3 to 6 display the test results when using KNN, SVM, LR, and MLP as classifiers, respectively. The best results were highlighted in bold and the second-best results were underlined in each column.

Table 3. Performance of the initial weights when using k-nearest neighbor (KNN)

Method	Metrics			
	Accuracy	Precision	Recall	F-Score
NW	<u>0.9592</u>	<u>0.9585</u>	0.9631	<u>0.9608</u>
TF-IDF	0.9292	0.9243	0.9406	0.9324
ReF	0.9558	0.9582	0.9567	0.9574
ECCD	0.9500	0.9351	**0.9711**	0.9528
EMSP	0.9558	0.9524	0.9631	0.9577
JOWM-IW	**0.9658**	**0.9694**	<u>0.9647</u>	**0.9670**

Table 4. Performance of the initial weights when using support vector machine (SVM)

Method	Metrics			
	Accuracy	Precision	Recall	F-Score
NW	0.9492	0.9391	**0.9647**	0.9517
TF-IDF	0.9442	0.9427	0.9502	0.9464
ReF	0.9458	0.9359	<u>0.9615</u>	0.9485
ECCD	0.9308	0.9441	0.9213	0.9326
EMSP	<u>0.9533</u>	<u>0.9465</u>	**0.9647**	<u>0.9555</u>
JOWM-IW	**0.9542**	**0.9479**	**0.9647**	**0.9562**

Table 5. Performance of the initial weights when using logistic regression (LR)

Method	Metrics			
	Accuracy	Precision	Recall	F-Score
NW	0.9508	0.9462	0.9599	0.9530
TF-IDF	0.9425	0.9453	0.9438	0.9446
ReF	0.9492	0.9418	0.9615	0.9515
ECCD	0.9200	0.9341	0.9101	0.9220
EMSP	0.9392	0.9379	0.9454	0.9416
JOWM-IW	**0.9550**	**0.9480**	**0.9663**	**0.9571**

Table 6. Performance of the initial weights when using multilayer perceptron (MLP)

Method	Metrics			
	Accuracy	Precision	Recall	F-Score
NW	0.9533	0.9479	0.9631	0.9554
TF-IDF	0.9517	0.9491	0.9583	0.9537
ReF	**0.9650**	**0.9560**	**0.9775**	**0.9667**
ECCD	0.8992	0.9183	0.8844	0.9011
EMSP	0.9192	0.9283	0.9149	0.9216
JOWM-IW	0.9367	0.9390	0.9390	0.9390

From the four tables, the following observations were made. First, when using the KNN, SVM, and LR classifiers, JOWM-IW achieves better performance than the other methods. When using the MLP classifier, ReF performs best, and JOWM-IW outperforms ECCD and EMSP. Therefore, on the whole, JOWM-IW has the best performance. The initial weights already achieved good performance because JOWM uses ML models to calculate the initial weight of each feature and integrates the results of multiple models. Second, TF-IDF, ReF, ECCD, and EMSP exhibited worse performance than NW in some cases, and JOWM-IW achieved worse performance than NW when using the MLP classifier. There are two main reasons. First, TF-IDF, ReF, ECCD, and EMSP estimate the importance of each feature solely on the statistical measures of data, and such importance may not be well suited to the classification models. Second, IG was used to select a small number of the most important features; thus, the selected features are all important, and it is challenging to distinguish their importance.

5.4 Performance Improvement of Weight-Mapping and Joint Parameter Optimization

Based on JOWM-IW, this section presents the evaluation of the effectiveness of the weight-mapping and joint parameter optimization and compares JOWM-IO and JOWM against it. The parameters in the joint optimization algorithm are set as follows. The population size is 40 (10 times the number of variables). The scaling factor is 0.5, and the crossover probability is 0.3. The maximal generation is set to 30. In addition, γ ranges from 10^{-6} to 10 for all weight-mapping functions. Moreover, β is in the range of 10^{-6} to 10 for the exponential function (EF) and hyperbolic function (HF). Moreover, α ranges from 0 to 10 for the power function (PF), from $1+10^{-6}$ to 10 for the EF, and from 10^{-6} to 10 for the logarithmic function (LF), HF, and S-shaped curve function (SF). Tables 7 to 10 list the test results when using KNN, SVM, LR, and MLP as classifiers, respectively.

Table 7. Performance improvement of weight-mapping when using k-nearest neighbor (KNN)

Method	Weight-mapping		Metrics			
	Mapping function	Optimized parameters	Accuracy	Precision	Recall	F-Score
JOWM-IW	-	-	0.9658	0.9694	0.9647	0.9670
JOWM-IO	PF	$\alpha = 3.109,\ \gamma = 5.390$	0.9692	0.9675	0.9723	0.9699
	EF	$\alpha = 3.070,\ \beta = 0.083,\ \gamma = 2.014$	0.9708	0.9692	0.9739	0.9715
	LF	$\alpha = 0.00001,\ \gamma = 8.541$	**0.9742**	**0.9739**	**0.9755**	**0.9747**
	HF	$\alpha = 8.504,\ \beta = 0.056,\ \gamma = 8.968$	0.9725	0.9708	**0.9755**	0.9731
	SF	$\alpha = 0.068,\ \gamma = 3.829$	0.9712	0.9723	0.9734	0.9726

Table 8. Performance improvement of weight-mapping when using support vector machine (SVM)

Method	Weight-mapping		Metrics			
	Mapping function	Optimized parameters	Accuracy	Precision	Recall	F-Score
JOWM-IW	-	-	0.9542	0.9479	0.9647	0.9562
JOWM-IO	PF	$\alpha = 0.161,\ \gamma = 0.665$	0.9700	0.9691	0.9711	0.9707
JOWM		$\alpha = 0.034,\ \gamma = 0.335,\ \delta = 12.57$	**0.9725**	**0.9739**	**0.9723**	**0.9731**
JOWM-IO	EF	$\alpha = 2.792,\ \beta = 6.813,\ \gamma = 9.366$	0.9700	0.9722	0.9690	0.9706
JOWM		$\alpha = 9.152,\ \beta = 2.410,\ \gamma = 4.604,\ \delta = 0.360$	0.9708	0.9738	0.9691	0.9714
JOWM-IO	LF	$\alpha = 7.870,\ \gamma = 1.423$	0.9708	0.9710	0.9706	0.9714
JOWM		$\alpha = 6.201,\ \gamma = 0.211,\ \delta = 13.56$	0.9717	0.9716	0.9713	0.9720
JOWM-IO	HF	$\alpha = 6.532,\ \beta = 9.035,\ \gamma = 0.892$	0.9700	0.9702	0.9705	0.9706
JOWM		$\alpha = 3.819,\ \beta = 4.937,\ \gamma = 0.495,\ \delta = 4.919$	0.9708	0.9707	0.9709	0.9715
JOWM-IO	SF	$\alpha = 3.929,\ \gamma = 1.841$	0.9692	0.9706	0.9693	0.9698
JOWM		$\alpha = 2.888,\ \gamma = 2.263,\ \delta = 0.910$	0.9700	0.9691	**0.9723**	0.9707

Table 9. Performance improvement of weight-mapping when using logistic regression (LR)

Method	Weight-mapping		Metrics			
	Mapping function	Optimized parameters	Accuracy	Precision	Recall	F-Score
JOWM-IW	-	-	0.9550	0.9480	0.9603	0.9571
JOWM-IO	PF	$\alpha = 0.070,\ \gamma = 2.190$	0.9717	0.9754	0.9696	0.9722
JOWM		$\alpha = 0.033,\ \gamma = 2.051,\ \delta = 1.028$	**0.9725**	**0.9770**	0.9690	**0.9730**
JOWM-IO	EF	$\alpha = 2.108,\ \beta = 1.107,\ \gamma = 2.963$	0.9675	0.9705	0.9657	0.9681
JOWM		$\alpha = 1.000,\ \beta = 4.019,\ \gamma = 0.647,\ \delta = 10.32$	0.9692	0.9691	**0.9706**	0.9698
JOWM-IO	LF	$\alpha = 2.523\ ,\ \gamma = 2.427$	0.9667	0.9713	0.9615	0.9664
JOWM		$\alpha = 5.999,\ \gamma = 3.579,\ \delta = 0.470$	0.9685	0.9713	0.9632	0.9673
JOWM-IO	HF	$\alpha = 8.205,\ \beta = 6.394,\ \gamma = 1.656$	0.9683	0.9714	0.9649	0.9681
JOWM		$\alpha = 8.431,\ \beta = 5.535,\ \gamma = 0.994,\ \delta = 1.416$	0.9708	0.9722	0.9704	0.9714
JOWM-IO	SF	$\alpha = 2.959,\ \gamma = 2.422$	0.9675	0.9705	0.9657	0.9681
JOWM		$\alpha = 2.907,\ \gamma = 0.725,\ \delta = 11.20$	0.9693	0.9706	0.9690	0.9697

Table 10. Performance improvement of weight-mapping when using multilayer perceptron (MLP)

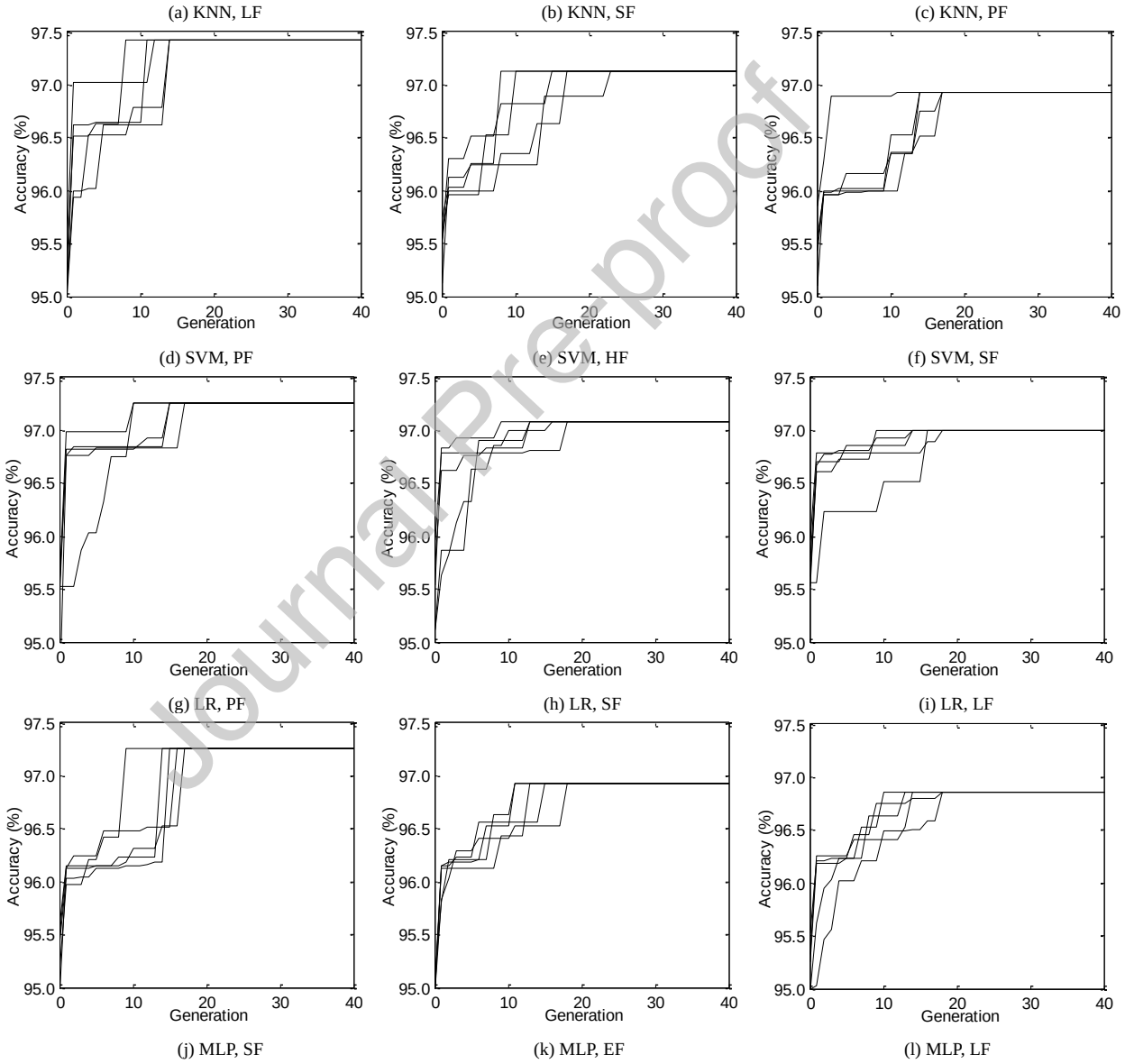
Method	Weight-mapping		Metrics			
	Mapping function	Optimized parameters	Accuracy	Precision	Recall	F-Score
JOWM-IW	-	-	0.9367	0.9390	0.9390	0.9390
JOWM-IO	PF	$\alpha = 0.076,\ \gamma = 8.151$	0.9683	0.9606	0.9791	0.9698
JOWM		$\alpha = 1.956,\ \gamma = 4.695,\ \delta = 0.024$	0.9750	0.9724	0.9788	0.9756
JOWM-IO	EF	$\alpha = 3.176,\ \beta = 2.891,\ \gamma = 9.647$	0.9717	0.9707	0.9739	0.9723
JOWM		$\alpha = 5.112,\ \beta = 0.654,\ \gamma = 1.662,\ \delta = 0.046$	0.9767	0.9756	0.9788	0.9772
JOWM-IO	LF	$\alpha = 2.104,\ \gamma = 9.771$	0.9683	0.9592	0.9807	0.9698
JOWM		$\alpha = 3.125,\ \gamma = 6.907,\ \delta = 0.018$	0.9700	0.9769	0.9641	0.9704
JOWM-IO	HF	$\alpha = 0.603,\ \beta = 7.919,\ \gamma = 7.645$	0.9700	0.9637	0.9791	0.9713
JOWM		$\alpha = 8.339,\ \beta = 5.631,\ \gamma = 1.902,\ \delta = 0.048$	0.9783	0.9757	**0.9821**	0.9789
JOWM-IO	SF	$\alpha = 0.635,\ \gamma = 8.469$	0.9650	0.9633	0.9695	0.9664
JOWM		$\alpha = 6.268,\ \gamma = 6.567,\ \delta = 0.021$	**0.9808**	**0.9805**	**0.9821**	**0.9813**

The results in the four tables reveal the following observations. First, no matter which classifier or which mapping function is used, JOWM-IO achieves better performance than JOWM-IW, and JOWM achieves better performance than JOWM-IO. This indicates that both weight-mapping and joint parameter optimization effectively improve the performance of the four classifiers. Second, for different classifiers, the best mapping functions are different. For example, PF is the most appropriate weight-mapping function for the SVM classifier, and SF is the most appropriate weight-mapping function for the MLP classifier. In a practical application, one can take a

certain proportion of samples from the training set as a verification set and choose the most appropriate weight-mapping function according to the performance on the verification set. Third, when using the MLP classifier, JOWM-IW performs worse than ReF (see Table 6), but the performance of JOWM greatly exceeds that of ReF.

5.5 Efficiency of the Joint Optimization Algorithm

The joint optimization algorithm is based on the DE algorithm, which is a population-based iterative optimization algorithm. This subsection aims to test the efficiency of the joint optimization algorithm (i.e., whether the algorithm can converge quickly). For each classifier, three mapping functions were chosen to perform the tests, whose performance is first, third, and fifth among the five mapping functions. During the optimization process, the current maximal fitness value of each generation was recorded. Each test was repeated five times. Here, the maximal generation was set to 40. Except for the maximal generation, the setting of the other parameters was completely consistent with Section 5.4. The results are displayed in Figure 6. Joint optimization has high efficiency and can converge in 25 generations or fewer.



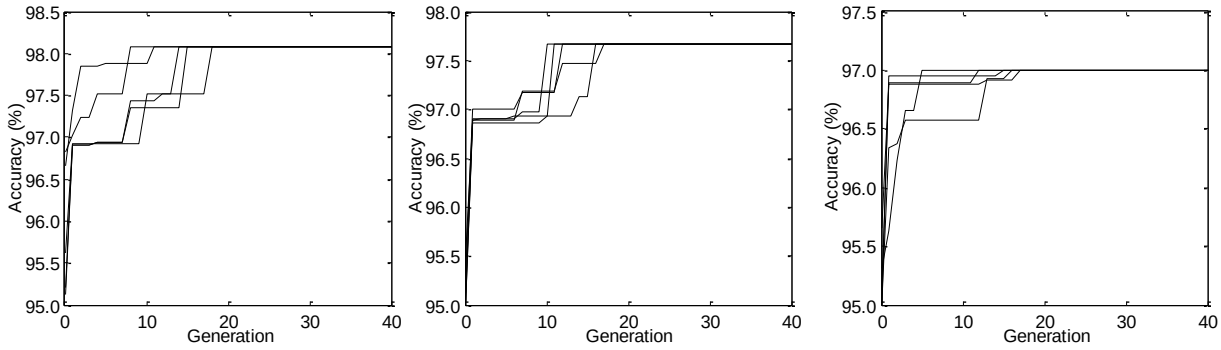


Figure 6. Optimization efficiency

Next, the training time of the model is discussed. The training of the model is carried out offline, and only needs to be trained once and then used repeatedly. Therefore, the training time of the model is not particularly important. For each individual, five classifiers need to be trained to calculate the fitness value (due to the adoption of the five-fold cross-validation). However, (i) the number of features after feature selection is not too large, so the training time of a single classifier is very short; (ii) as only four parameters need to be optimized, the size of the population is small; and (iii) the DE algorithm has a fast convergence speed, so not many iterations are required. Therefore, the entire training time of the model is not too long.

In the above tests (only a common PC was used for the tests, and its configuration is shown in Section 5.1), when the SVM classifier and PF mapping function were used, the training time of the model was 2.1 hours; and when the MLP classifier and HF mapping function were used, the training of the model took 5.8 hours. Considering that computing fitness value for each individual does not interfere with each other, if one wants to shorten the training time of the model, he can use multi-threading or multi-machine parallel computing [45], [46] to calculate the fitness values of the individuals.

5.6 Performance Comparison with Weight-Unaware Classifiers

This subsection compares the four weight-aware classifiers equipped with the proposed feature weighting method against five common weight-unaware classifiers: Gaussian NB, multinomial NB, LDA, QDA, and RF. To make a fairer comparison, default parameters were used for all classifiers; thus, the feature weighting method is JOWM-IO. In addition, for each weight-aware classifier, the weight-mapping function that provides the worst performance was used to make the comparison. The results are listed in Table 11. To be more convincing, the table also lists the performance of the weight-aware classifiers when using the NW method (i.e., no feature weighting method).

Table 11. Performance comparison with weight-unaware classifiers

Classifier		Feature weighting		Optimized parameters	Metrics			
Category	Name	Weighting method	Mapping function		Accuracy	Precision	Recall	F-Score
Weight-unaware	Gaussian NB	-	-	-	0.6680	0.6020	0.9890	0.7490
	Multinomial NB	-	-	-	0.9150	0.9199	0.9139	0.9169
	LDA	-	-	-	0.9491	0.9411	0.9610	0.9510
	QDA	-	-	-	0.9291	0.9468	0.9066	0.9262
	RF	-	-	-	0.9525	0.9574	0.9496	0.9535
Weight-aware	KNN	NW	-	-	0.9592	0.9585	0.9631	0.9608
		JOWM-IO	PF	$\alpha = 3.109,\ \gamma = 5.390$	0.9692	0.9675	0.9723	0.9699
	SVM	NW	-	-	0.9492	0.9391	0.9647	0.9517

		JOWM-IO	SF	$\alpha = 3.929,\ \gamma = 1.841$	0.9692	0.9706	0.9693	0.9698
	LR	NW	-	-	0.9508	0.9462	0.9599	0.9530
		JOWM-IO	LF	$\alpha = 2.523,\ \gamma = 2.427$	0.9667	0.9713	0.9615	0.9664
	MLP	NW	-	-	0.9533	0.9479	0.9631	0.9554
		JOWM-IO	SF	$\alpha = 0.635,\ \gamma = 8.469$	0.9650	0.9633	0.9695	0.9664

Table 11 reveals that RF has the best performance among the five weight-unaware classifiers. If no feature weighting method is used, KNN and MLP have better performance than RF, and SVM and LR have worse performance than RF. However, if equipped with the proposed feature weighting method, the performance of SVM and LR surpassed that of RF. Therefore, the proposed feature weighting method makes the weight-aware classifiers more competitive.

6. Conclusion

A novel feature weighting method (JOWM) and an Android malware detection scheme (JOWMDroid) were proposed in this paper. The JOWMDroid scheme uses the static analysis method and includes the following five steps. First, the original features of eight categories were extracted from the APK files. Then, IG was used to select a certain number of the most important features. Next, an initial weight was calculated for each selected feature via three ML models. Five weight-mapping functions were designed to map the initial weight to the final weight. Finally, the parameters of the weight-mapping function and classifier were jointly optimized using the DE algorithm.

This paper also compares JOWM with four state-of-the-art feature weighting methods and compares the performance of the four weight-aware and five weight-unaware classifiers. The experimental results indicate that the initial weights already achieved good performance. Moreover, weight-mapping and joint parameter optimization both effectively improved the accuracy of malware detection. The proposed method outperforms the four state-of-the-art methods. In addition, the joint optimization can quickly converge to a group of excellent parameters. Further, when equipped with the proposed feature weighting method, the weight-aware classifiers all outperform the weight-unaware classifiers.

In the future, the authors will consider the correlation between features. Moreover, they will study how to build joint features to improve the detection accuracy of Android malware.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (No. 61202366), and the Natural Science Foundation of Guangdong Province (No. 2018A030313438, 2018A030313889).

References

[1] Statcounter, “Mobile Operating System Market Share Worldwide,” [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[2] R. Brandom, “There are now 2.5 billion active Android devices,” [Online]. Available: https://www.theverge.com/2019/5/7/18528297/google-io-2019-android-devices-play-store-total-number-statistic-keynote.

[3] G. Meng, M. Patrick, Y. Xue, Y. Liu, and J. Zhang, “Securing Android App markets via modeling and predicting malware spread between markets,” *IEEE Transactions on Information Forensics and Security*, vol.

14, no. 7, pp. 1944–1959, 2019.

[4] Keen Security Lab of Tencent, "Android application security white paper 2018," [Online]. Available: https://paper.seebug.org/953.

[5] Symantec, "Internet security threat report," *Cupertino, CA, USA, Tech.Rep.*, vol. 20, 2015.

[6] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[7] D. O. Sahin, O. E. Kural, S. Akleylek, and E. Kilic, "New results on permission based static analysis for android malware," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, 2018, pp. 1–4.

[8] A. A. Sahal, S. Alam, and I. Sogukpinar, "Mining and detection of android malware based on permissions," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, 2018, pp. 264–268.

[9] Z. Xiong, T. Guo, Q. Zhang, Y. Cheng, and K. Xu, "Android malware detection methods based on the combination of clustering and classification," in *12th International Conference on Network and System Security (NSS), 2018*, pp. 411–422.

[10] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting android malware," *Computer Networks*, vol. 167, Article 107026, 2020.

[11] O. Olukoya, L. Mackenzie, and I. Omoronyia, "Towards using unstructured user input request for malware detection," *Computers & Security*, vol. 93, Article 101783, 2020.

[12] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1231–1245, 2019.

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

[14] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, "Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting Android malware with system call sequences," *IET Information Security*, vol. 11, no. 1, pp. 8–15, 2017.

[15] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: Detecting Android malware by ORGB analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, 2020.

[16] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.

[17] Z. Wang, K. Li, Y. Hu, A. Fukuda, and W. Kong, "Multilevel permission extraction in Android applications for malware detection," in *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 2019, pp. 1–5.

[18] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of Android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, 2018.

[19] A. K. Singh, C. D. Jaidhar, and M. A. A. Kumara, "Experimental analysis of Android malware detection based on combinations of permissions and API-calls," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 3, pp. 209–218, 2019.

[20] A. Appice, G. Andresini, and D. Malerba, "Clustering-aided multi-view classification: A case study on Android malware detection," *Journal of Intelligent Information Systems*, vol. 55, pp. 1-26, 2020.

[21] Y. Liu, K. Guo, X. Huang, Z. Zhou, and Y. Zhang, "Detecting Android malwares with high-efficient hybrid analyzing methods," *Mobile Information Systems*, vol. 2018, Article 1649703, 2018.

[22] A. Kumar, K. S. Kuppusamy, and G. Aghila, "FAMOUS: Forensic analysis of mobile devices using scoring of

application permissions," *Future Generation Computer Systems*, vol. 83, pp. 158–172, 2018.

[23] M. V. Varsha, P. Vinod, and K. A. Dhanya, "Identification of malicious Android App using manifest and opcode features," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 2, pp. 125–138, 2017.

[24] M. Yang, S. Wang, Z. Ling, Y. Liu, and Z. Ni, "Detection of malicious behavior in Android Apps through API calls and permission uses analysis," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 19, Article e4172, 2017.

[25] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, 2019.

[26] M. Z. Mas'ud, S. Sahib, M. F. Abdollah, S. R. Selamat, and C. Y. Huoy, "A comparative study on feature selection method for N-gram mobile malware detection," *International Journal of Network Security*, vol. 19, no. 5, pp. 727–733, 2017.

[27] 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.

[28] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proceedings 2014 Network and Distributed System Security Symposium (NDSS)*, 2014.

[29] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based Android malware detection using real devices," *Computers & Security*, vol. 89, Article 101663, 2020.

[30] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[31] A. Bhattacharya, R. T. Goswami, and K. Mukherjee, "A feature selection technique based on rough set and improvised PSO algorithm (PSORS-FS) for permission based detection of Android malwares," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 7, pp. 1893–1907, 2019.

[32] 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.

[33] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A Multi-view context-aware approach to Android malware detection and malicious code localization," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1222–1274, 2018.

[34] H.-T. Nguyen, Q.-D. Ngo, and V.-H. Le, "A novel graph-based approach for IoT botnet detection," *International Journal of Information Security*, 2019. https://doi.org/10.1007/s10207-019-00475-6.

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

[36] Y. Liu, L. Zhang, and X. Huang, "Using G features to improve the efficiency of function call graph based Android malware detection," *Wireless Personal Communications*, vol. 103, no. 4, pp. 2947–2955, 2018.

[37] Z. Xu, K. Ren, and F. Song, "Android malware family classification and characterization using CFG and DFG," in *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2019, pp. 49–56.

[38] H. Naeem, B. Guo, F. Ullah, and M. R. Naeem, "A Cross-platform malware variant classification based on image representation," *KSII Transactions on Internet and Information Systems*, vol. 13, no. 7, pp. 3756–3777, 2019.

[39] Androguard Team, "Androguard," [Online]. Available: https://github.com/androguard/androguard.

[40] Jason Brownlee, "How to perform feature selection with machine learning data in weka," [Online]. Available: https://machinelearningmastery.com/perform-feature-selection-machine-learning-data-weka/.

[41] "Scikit-learn." [Online]. Available: http://scikit-learn.org/stable/.

[42] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*,

2017, pp. 252–276.

[43] Google, “Google play store,” [Online]. Available: https://play.google.com/store/apps?hl=en.

[44] APKPure Team, “APKPure.com,” [Online]. Available: https://apkpure.com/cn/.

[45] Z. Xiong and K. Xu, “Lightweight job submission and file sharing schemes for a teaching ecosystem for parallel computing courses,” *Journal of Ambient Intelligence and Humanized Computing*, 10 February 2020, https://doi.org/10.1007/s12652-020-01695-8.

[46] “Parallel processing and multiprocessing in python,” [Online]. Available: https://wiki.python.org/moin/ParallelProcessing.