

Research Article

Computing Adaptive Feature Weights with PSO to Improve Android Malware Detection

Yanping Xu,¹ Chunhua Wu,¹ Kangfeng Zheng,¹ Xu Wang,¹ Xinxin Niu,¹ and Tianliang Lu²

¹*School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing, China*

²*School of Information Technology and Network Security, People's Public Security University of China, Beijing 100038, China*

Correspondence should be addressed to Yanping Xu; xyp_xyp@126.com

Academic Editor: Pedro Peris-Lopez

Copyright © 2017 Yanping Xu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Android malware detection is a complex and crucial issue. In this paper, we propose a malware detection model using a support vector machine (SVM) method based on feature weights that are computed by information gain (IG) and particle swarm optimization (PSO) algorithms. The IG weights are evaluated based on the relevance between features and class labels, and the PSO weights are adaptively calculated to result in the best fitness (the performance of the SVM classification model). Moreover, to overcome the defects of basic PSO, we propose a new adaptive inertia weight method called fitness-based and chaotic adaptive inertia weight-PSO (FCAIW-PSO) that improves on basic PSO and is based on the fitness and a chaotic term. The goal is to assign suitable weights to the features to ensure the best Android malware detection performance. The results of experiments indicate that the IG weights and PSO weights both improve the performance of SVM and that the performance of the PSO weights is better than that of the IG weights.

1. Introduction

Malicious Android applications are pervasive in smart mobile devices. The most notorious type of malicious application involves privacy theft. These applications steal sensitive private information such as contact lists, text messages, photos, geolocations, and users' accounts through various means, including accessing information without permission and hijacking message transmissions through networks. Therefore, improving the ability to detect malware on mobile devices is of paramount importance.

Android malware detection is a crucial and challenging issue and has attracted the attention of thousands of researchers. The works can be divided into three categories: static analysis, dynamic analysis, and hybrid analysis [1]. The analyzed features focus on permissions [2, 3], APIs [4], and combined features [5–8]. When the number of features is large, feature selection methods are used to reduce the quantity and improve the performance and efficiency [8–10]. Most previous studies have treated all features as equally important and represented the samples with Boolean values that simply indicate whether a feature appears; however, this

approach does not consider the relative influence of various features in malware detection [2, 3, 5]. However, not all features are equal when evaluating their similarity to Android features. For example, SMS-related permissions and APIs are often used in malicious applications but are used less often in benign ones, while Internet-related permissions and APIs are often employed by both benign and malicious applications. Therefore, SMS-related information has a strong influence on Android malware detection and classification, but Internet-related features have only a weak influence. To overcome this problem, a suitable feature weight should be computed for each feature.

Feature weight methods were originally proposed in document categorization to represent the relative importance of features. For example, term frequency-inverse document frequency (TF-IDF) term weights are considered the attribute values of documents in many information retrieval models [11]. Overall, feature weighting methods can generally be divided into three types: term frequency methods [12], feature rank methods [13], and self-adjustment methods [14, 15]. Term frequency represents the capacity of features for expressing document content, and it is computed by the

number of times a feature is mentioned in a set of documents. Feature rank methods, such as information gain (IG) and Chi-square (CHI), calculate the mutual information between features and labels and assign high values to relevant features and small values to less relevant features. Self-adjustment methods utilize machine learning models and assign feature weights based on the best performance of the classifiers. For a given dataset, feature weights can change the distribution of the samples, which makes similar samples more compact and widens the distribution of dissimilar samples [16].

However, Android source code is different from words in documents. Some important features, such as SMS-related permissions, may appear several times in one application's source code but may have a low frequency overall in the source code files. In contrast, high frequency features such as `toString()` often appear thousands of times in both malicious and benign applications; therefore, they do not work well as a distinguishing feature. Consequently, in Android source code, term frequency methods are not as good as feature rank methods for determining feature weights. In addition, because there are more than ten thousand features, some of which appear over a thousand times, calculating the frequency statistics for all features is computationally expensive. Moreover, the results of experiments conducted in [8] revealed that term frequency representation performs worse than binary representation for Android malware detection. Therefore, in this work, we do not compare the performance of term frequency weights with the performances of other weight methods.

Particle swarm optimization (PSO) is an optimization method motivated by the intelligent collective behavior of a swarm. It uses a set of particles moving through the search space to achieve the best fitness of an optimization objective [17]. Based on this, PSO can be used to adaptively compute the feature weights, where PSO is wrapped into the machine learning classifiers. The positions of the particles in the solutions are taken as the weights, and the fitness is calculated based on the accuracy of the machine learning classifiers. The particles search in the weight space to generate the weight for each feature. Then, the weights are multiplied with the features to form a new mapped feature space. Next, the new feature space is input to the model of the machine learning classifiers to determine their fitness (accuracy). However, basic PSO has some defects; for example, the swarm can easily converge prematurely and its local search capability is poor. Therefore, we propose a new adaptive inertia weight method for PSO to improve its particle searching ability [17, 18].

In this work, we propose an Android malware detection model and describe the details of malware detection using feature weights computed separately by IG and PSO. To solve the problems of PSO, we propose an adaptive inertia weight method for PSO that is based on their fitness and a chaotic sequence, which is used to assign suitable weights for the features. The values of the weights are restricted to the range $[0, 1]$. The larger the weight is, the more important the feature is for malware detection [19]. Our research makes several significant contributions to the field of Android malware static detection:

- (1) We propose a model for Android malware detection based on a support vector machine (SVM) classification model using IG and PSO feature weights. The feature weights are multiplied by the features to make similar samples more compact and dissimilar samples more distant; therefore, they simplify the classification between malware and benign applications.
- (2) Based on the relevance between features and class labels, IG is used not only to select a specified number of features but also to evaluate feature weights. The results indicate that IG weights have little effect on performance.
- (3) We propose a new adaptive inertia weight method called fitness-based and chaotic adaptive inertia weight-PSO (FCAIW-PSO) for basic PSO that is based on both the fitness and a chaotic term to improve the particle searching ability. The new method can assign suitable weights for Android features and ensure the highest detection accuracy.

The remainder of this paper is organized as follows. Section 2 reviews related works on Android static malware detection and feature weight analysis. Section 3 presents details of the research methodology. Section 4 introduces and discusses the experiments, and Section 5 provides conclusions.

2. Related Work

Numerous studies have been conducted in the field of Android malware detection. Static analysis is a popular method for Android malware detection because of its simplicity and efficiency [1]. In addition, assigning appropriate feature weights is an important theme in our study. This section reviews related works on static feature detection and feature weight selection.

2.1. Static Detection. To detect malicious applications outside mobile devices, multiple methods have been proposed to extract and analyze features from decompiled source code [20]. Studies on static feature detection include several aspects such as which features are extracted, what types of methods are used to detect features, and how to detect malware with machine learning and other methods. In this section, we include several of the most recent and significant results.

Permission-induced risk in Android applications is explored in [21], which uses feature ranking methods to rank permissions with respect to their risks. Then, machine learning models such as SVM, logistic regression (LR), decision tree (DT), and random forest (RF) are employed as classification algorithms to detect malware that requests risky permissions. The empirical results show that these methods provide satisfactory performance and achieve a detection rate as high as 94.62%. The authors of [8] propose using a probabilistic discriminative model for Android malware detection using API calls and permissions extracted from decompiled source code as combined features. Probabilistic discriminative models such as regularized logistic regression

(RLR), naïve Bayes with informative priors (PNB), hierarchical mixture of naïve Bayes (HMNB), and 2-class naïve Bayes with prior (2-PNB) are then used to classify Android applications as malware or benign. Feature selection methods are applied to improve the performance.

A novel semantic-based approach proposed in [4], called DroidSIFT, detects malware using weighted contextual dependency graphs and introduces graph similarity metrics to reveal homogeneous application behaviors. Experiments show that DroidSIFT's method can achieve high accuracy on both known and zero-day malicious applications. In [22], the authors extracted data-flow features for users' trigger sensitive APIs and built a program called TriggerMetric that captures the data dependence relations between a user's actions and sensitive operations. The values of TriggerMetric reflect the degree of sensitive operations that are triggered or intended by the users. Their method also discovers new malicious applications in the Google Play store that cannot be detected by virus scanning tools.

In previous works, features have typically been regarded as equally important for malware detection, although some permissions and APIs are very relevant to the application domains [21]. In this work, we also utilize static analysis to discriminate between malware and benign applications. However, we do not believe that all features provide an equal amount of information for classification. For example, SMS-related permissions and functions are frequently used in malware, while Internet-related permissions and functions appear frequently in both types of applications. Consequently, we use feature weights to indicate the relative importance of features for classification.

2.2. Feature Weights. Feature weights have been studied for both text and anomaly detection; they can change the distribution of the sample dataset by making similar class samples more compact. Several feature weight methods such as term frequency methods [12], feature rank methods [13], and self-adjustment methods [14, 15] have been proposed and utilized in practice. Choosing appropriate feature weights can improve classification.

TF-IDF is a feature weight method that estimates the frequency of terms in the feature space. Four feature weight methods were studied in [12], that is, TF-IDF, TF-CRF, TF-OddsRatio, and TF-CHI, to calculate feature values for text categorization. The results of experiments indicated that TF-CHI is the most effective for use with SVM and results in high accuracy. The authors of [11] applied TF-IDF weights in a probabilistic retrieval model to make relevance decisions for documents, while [23] used TF-IDF as a term weighting method for text representation. The results indicate that TF-IDF performs well for Chinese information retrieval and English information retrieval.

Feature ranking methods, which are based on statistics and information theory, can be considered a category of feature selection methods. These methods rank features in descending order according to the relevance between features and labels [24]. The feature weighting method proposed in [25], which is based on a χ^2 statistical test, was used with the

k -nearest neighbors (kNN) classifier. Forty-four out of forty-five experiments demonstrated that the proposed weighting process achieved a good performance on a dataset that included many irrelevant features. In [13], a new fast feature weighting algorithm was proposed based on feature ranking methods. The ranking methods involved correlations, information theory, and distance between probability distributions. Then, the weighting schemes were combined with SVM and kNN network classification techniques. The experiment results indicated that the proposed feature weighting methods were fast and accurate with these classifiers. Feature rank methods are useful for determining the relative importance of individual features; however, they neglect the possible interactions between features.

Very different from the two types of methods discussed above, self-adjusting weighting mechanisms are integrated into machine learning classifiers to improve their performance. The self-adjusted feature weights adopt the best value based on the best performances of the subsequent classifiers. A fast iterative feature weight adjustment algorithm based on a linear-complexity centroid classifier for document categorization was presented in [14]. This study measured the discriminating power of each term to gradually adjust the weight of all features concurrently. Through experiments, the authors showed that their feature weight adjustment method improved the performance of the centroid-based classifier by 2–5%. In [15], a feature weight self-adjustment (FWSA) mechanism that worked with k -means was proposed to improve the clustering performance. The process of finding feature weights involved optimizing the k -means classification. The proposed mechanism illustrated several advantages both theoretically and through the experimental results.

For classification purposes, there is a considerable difference between the applicability of features for Android malware detection and text categorization. For example, TF-IDF is useless as a feature weight in Android malware detection. However, we can learn from the theories of feature rank methods and self-adjustment weight mechanisms used in text categorization to design feature weighting methods for Android malware detection. Therefore, both IG and PSO are used in our work because IG can rank features and evaluate weight using an IG value, while PSO can be integrated with a classifier to adaptively adjust the weights.

3. The Proposed Detection Model

The structure and process of Android malware detection with feature weights are depicted in Figure 1. First, permissions and functions are extracted as features from the Android dataset. Second, the dataset is represented as a matrix with the features. The feature weights are calculated with IG and PSO; then, the mapped matrix is obtained by the feature weights and the dataset matrix. As shown in Figure 1, the process is mainly divided into two branches. The top image in Figure 1 shows that the IG feature weights are computed by the IG method with the training dataset. In the training stage, the IG weights are multiplied by the training sample matrix to obtain a mapped matrix, which is then input into SVM to train

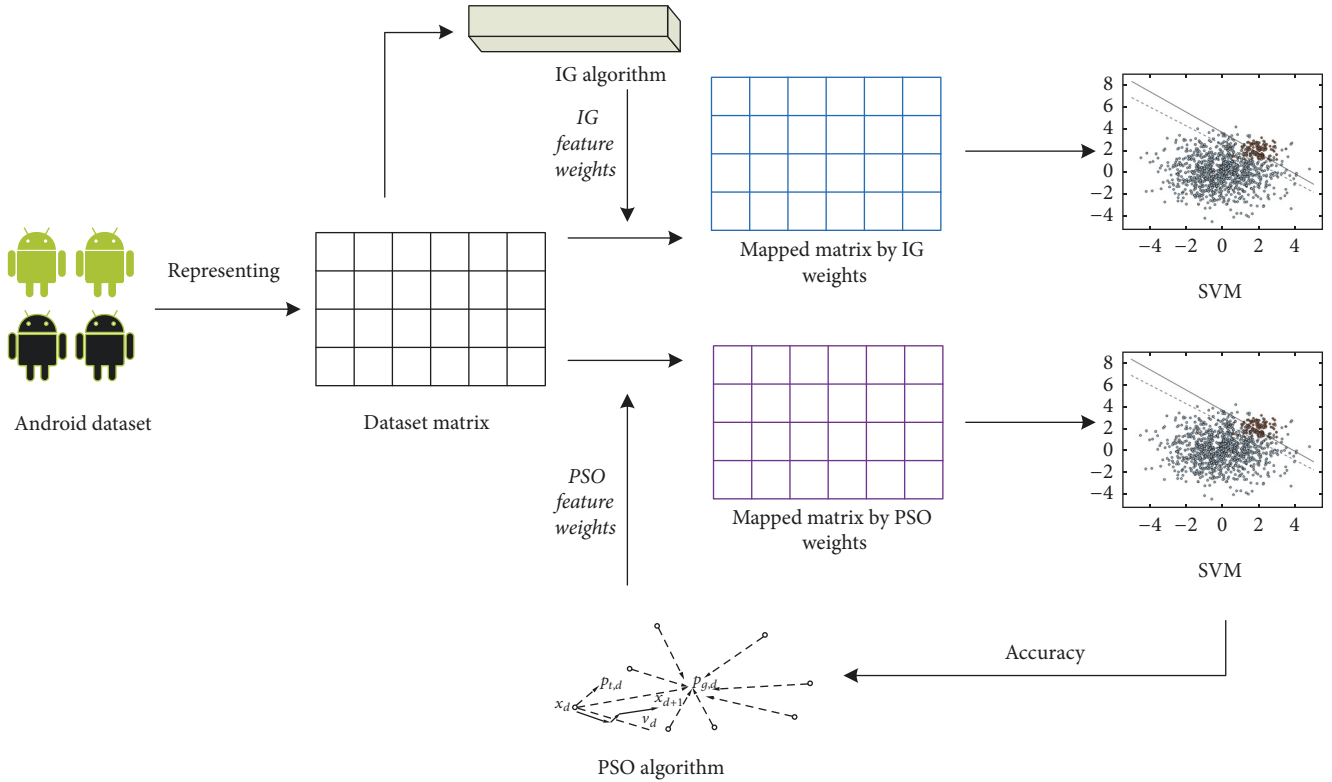


FIGURE 1: The structure of Android malware detection based on SVM with IG and PSO feature weights.

the classification model. In the testing stage, the IG weights are multiplied by the test sample matrix to obtain a mapped matrix, which is then input into the trained SVM model to identify the labels of the testing samples. The bottom image in Figure 1 shows how the PSO feature weights are calculated and work with the SVM classifier to detect malware. In the training stage, the PSO weights are initialized by a random matrix and are then adaptively updated to the best values fitted to the best SVM detection performance. In the testing stage, the PSO weights work similarly to the IG weights.

3.1. Information Gain. Information gain (IG), as a feature selection method, measures how much information a feature adds to the classification system. It computes an IG value for every feature. The larger the IG value is, the more information the feature contains. Therefore, the IG values can be viewed as IG feature weights that represent the importance of each feature.

Let x be the feature, $y_i \in y$ be one of the k sample class labels, and $IG(x)$ be the IG feature weight for feature x .

$$\begin{aligned} IG(x) &= H(y) - H(y|x) \\ &= -\sum_{i=1}^k p(y_i) \log(p(y_i)) \end{aligned}$$

$$\begin{aligned} &+ p(x) \sum_{i=1}^k p(y_i|x) \log(p(y_i|x)) \\ &+ p(\bar{x}) \sum_{i=1}^k p(y_i|\bar{x}) \log(p(y_i|\bar{x})), \end{aligned}$$

(1)

where $H(y)$ is the information entropy of a sample and $H(y|x)$ is the conditional entropy of feature x in the sample, which represents the information quantity (i.e., x exists or does not exist in the classification system) [26, 27].

The value of each IG weight is restricted to the range $[0, 1]$. Then, the representation value of a feature multiplied by its IG feature weight generates a new mapping value that changes the distribution of the original feature space and is used here to detect malware.

3.2. Particle Swarm Optimization

3.2.1. Basic Particle Swarm Optimization. PSO is an evolutionary optimization method based on a population (called a swarm). Each particle in the population has several parameters, including its current position, velocity, and best position. The particles in the swarm move around with velocity in the search space guided by their own previous best positions and best known positions of the entire swarm. When improved positions are discovered, those positions are used to guide

future movements of the swarm. The process is repeated until the optimal swarm positions are eventually discovered [28].

For an n -dimensional search space, the particle parameters are represented with n -dimensional vectors. The position and velocity of the i th particle are represented by $X_i = (x_{i1}, x_{i2}, \dots, x_{in})$ and $V_i = (v_{i1}, v_{i2}, \dots, v_{in})$, respectively. During the search process, at iteration $t + 1$, the position and velocity of the particles are updated according to

$$\begin{aligned} V_i(t+1) &= w \times V_i(t) + c_1 \times \text{rand}() \times (P_{pbest} - V_i(t)) \\ &\quad + c_2 \times \text{rand}() \times (P_{gbest} - V_i(t)), \end{aligned} \quad (2)$$

$$X_i(t+1) = X_i(t) + V_i(t+1),$$

where w is an inertia weight that ensures the convergence of the PSO algorithm, c_1 and c_2 are two learning factors that control the social and cognitive components of the swarm, $\text{rand}()$ consists of distinct random values within the range $[0, 1]$, P_{pbest} is the best local position of the particle itself, and P_{gbest} is the best previous position of all particles. The balance between global and local research can determine the results of the optimization algorithm. The inertia weight w is an important parameter that is set to a suitable value to strike a balance between global and local exploitation. Large w values promote global exploration, whereas small values promote local exploitation.

3.2.2. Inertia Weight Adaptation Strategies. Although PSO has a strong ability to solve optimization problems, it has several defects: the swarm can easily fall into premature convergence and its local search capability is poor [18]. To overcome these defects, various strategies have proposed to improve the inertia weight. In our work, we will introduce several significant types, as follows:

- (1) A constant value is used for the inertia weight within the range $[0.8, 1.2]$ to optimize the algorithm [29].
- (2) A random value is proposed for the inertia weight to improve PSO's optimization ability [30]:

$$w = 0.5 + \frac{\text{rand}()}{2}. \quad (3)$$

Using this method, the value of w is restricted to the range $[0.5, 1]$.

- (3) At the beginning of the PSO process, a higher global search ability promotes the exploration of new areas and maintains swarm diversity, whereas, later, a stronger local search can converge more quickly. Consequently, the inertia weight w is treated as a time-varying value that decreases linearly from a large value (w_{\max}) to a small value (w_{\min}) [17, 30]:

$$w(t) = \frac{\text{iter}_{\max} - t}{\text{iter}_{\max}} (w_{\max} - w_{\min}) + w_{\min}, \quad (4)$$

where iter_{\max} is the maximum number of iterations and t is the current iteration of the algorithm.

- (4) A chaotic sequence can show the ergodicity, randomness, and regularity of chaotic motion, and a chaotic sequence can traverse all the states within a certain range based on its own definition. Therefore, a chaotic sequence is used to enhance the diversity of the swarm search. A chaotic term is added to the linearly decreasing inertia weight. This chaotic term causes the inertia weight to change in sequential steps [31, 32]:

$$w(t) = \frac{\text{iter}_{\max} - t}{\text{iter}_{\max}} (w_{\max} - w_{\min}) + w_{\min} \times Z, \quad (5)$$

where Z is the chaotic term, which can be represented by different chaotic models such as Logistic, Lozi, Dissipative, and Arnold's Cat.

- (5) The time-varying inertia weight does not determine the situation of the particles in the search space, so they do not adapt to the movement of the swarm. Therefore, some adaptive inertia weight strategies are proposed. One adaptively changing inertia weight strategy uses the *Gompertz function*, which incorporates a double exponential function to select the adaptive inertia weight [33]:

$$\begin{aligned} w_i(t+1) &= \exp(-\exp(-R_i(t))), \\ R_i(t) &= |P_{gbest} - P_{pbest}| \times \frac{\text{iter}_{\max} - t}{\text{iter}_{\max}}, \end{aligned} \quad (6)$$

where $R_i(t)$ is computed as a performance index that depends on the particle's personal best position and the global best position of the swarm during the operation that moves the particles in each iteration. Using this method, the inertia weight is initialized to 0.4 and then gradually increases to approximately 1.

3.2.3. The Proposed Inertia Weight Strategy. In our work, the PSO algorithm is used to find the most appropriate feature weights. The positions of each particle are viewed as the feature weights, and the dimension of the positions is equal to the dimension of the features. The feature matrix is multiplied by the weight matrix to generate a mapped matrix. Then, the mapped matrix is input to the machine learning classifier to achieve an accuracy score, which is, in turn, treated as the fitness of the PSO algorithm. To achieve the highest fitness, it is necessary to search the suitable positions. In addition, the value of the position of each particle is restricted to the range $[0, 1]$.

The previous studies provide us with two ideas. (1) It is important to relate the inertia weight to the searching ability of the swarm so that an adaptive strategy can be used to improve inertia weight and maintain diversity. (2) Achieving the highest fitness, which depends on the best global optimal positions, is the goal of the PSO algorithm. To achieve this goal, a new method is proposed that improves the inertia weight based on the best fitness and is inspired by the idea presented in the previous section. Furthermore, incorporating a chaotic term can increase swarm diversity;

consequently, we consider adding a chaotic sequence to the adaptive strategy of the inertia weight. Based on the above discussion, the proposed fitness-based chaotic inertia weight strategy is introduced as follows:

- (1) When the local optimal value of one particle in the swarm is updated, this means that the particle has achieved its own optimum position and approached its global optimum position. Therefore, during the swarm iteration process, when the fitness of the particle i is updated to a larger value, we believe that the change is an indication that particle i is strong. Consequently, the strong particle i at iteration t is defined as follows:

$$\text{Strong}(i, t) = \begin{cases} 1, & \text{if } f(P(t)) > f(P(t-1)) \\ 0, & \text{if } f(P(t)) = f(P(t-1)), \end{cases} \quad (7)$$

where $P(t)$ is the local position of particle i at iteration t , $f(\cdot)$ is the fitness function of PSO, and $f(P(t))$ is the fitness of particle i at iteration t with the current local position.

- (2) The ratio of all the strong particles in the swarm during one iteration step is computed as follows:

$$N(t) = \frac{\sum_{i=1}^N \text{Strong}(i, t)}{N}, \quad (8)$$

where N is the number of the particles in the swarm. Therefore, the range of $N(t)$ is restricted to $[0, 1]$. According to the results of previous studies [17, 18, 30, 31], the higher the value of $N(t)$ is, the farther the swarm is from the global optimum value; therefore, a high value means that the global searching ability should be enhanced. In contrast, the smaller the value of $N(t)$ is, the closer the swarm is to the global optimum value; therefore, a small value means that the local searching ability should be enhanced.

- (3) Based on the analysis in the previous step, we create a relationship between the inertia weight value and the global/local search ability of the swarm and include the chaotic term [17, 31]. Our adaptive dynamic inertia weight strategy is defined as follows:

$$w(t) = \frac{\text{iter}_{\max} - t}{\text{iter}_{\max}} \times N(t) + w_{\min} \times Z, \quad (9)$$

where iter_{\max} is the maximum number of iterations, t is the current iteration of the algorithm, w_{\min} is the minimum inertia weight w , and Z is the chaotic term, as in formula (7). In the next section, we will use test environments to study the performance of the proposed fitness-based chaotic inertia weight strategy at given iterations during the course of a run. Furthermore, we compare other recent inertia weight strategies with our method.

3.3. Support Vector Machine. SVM aims to construct the best hyperplane in a sample space to act as a binary classifier [34]. The hyperplane function is defined as follows:

$$w \cdot x + b = 0, \quad (10)$$

where w is a normal vector and b is an intercept. The hyperplane separates the sample space into two parts: a positive part and a negative part. In general, there are an infinite number of hyperplanes, but the optimal hyperplane is achieved by finding the maximum margin distance to the nearest training data point of any class. The functional margin of the hyperplane to sample (x_i, y_i) is defined as

$$\hat{y}_i = y_i (w \cdot x_i + b). \quad (11)$$

In general, the larger the margin is, the lower the generalization error of the classifier is. Therefore, the goal of SVM is to find an optimal hyperplane that has the biggest margin distance and achieves a high classification accuracy on the test samples.

Most of the time, the sample space is not linearly separable; consequently, linear SVM cannot solve the problem. Therefore, kernel functions are defined that can map the original low-dimensional space to a high-dimensional space. Then, the hyperplane is constructed in this high-dimensional space, which can easily separate the sample space. Common kernel functions include linear, polynomial, Gaussian, and sigmoid [35]. In this work, we use a Gaussian kernel function based on the feature distribution characteristic.

However, there are some noise features in the sample set that usually have a great impact on the correct classification. To solve this problem, a slack variable ζ_i and a penalty factor C are presented. Finally, the original optimal problem of the nonlinear SVM becomes as follows:

$$\begin{aligned} \min_{w, b, \zeta} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \zeta_i \\ \text{s.t.} \quad & y_i (w \cdot x_i + b) \geq 1 - \zeta_i, \quad i = 1, 2, \dots, N, \\ & \zeta_i \geq 0, \quad i = 1, 2, \dots, N. \end{aligned} \quad (12)$$

Its dual is

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^N \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N, \end{aligned} \quad (13)$$

where $K(x_i, x_j)$ is the kernel function. The optimum solution of α^* is $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_N^*)^T$.

Then, optimum solutions for w^* and b^* are

$$\begin{aligned} w^* &= \sum_{i=1}^N \alpha_i^* y_i, \\ b^* &= y_i - \sum_{i=1}^N \alpha_i^* y_i K(x_i, x_j). \end{aligned} \quad (14)$$

TABLE 1: Confusion matrix.

	Positive	Negative
True	True positive (TP)	False negative (FN)
False	False positive (FP)	True negative (TN)

Finally, the hyperplane is

$$w^* \cdot x + b^* = 0 \quad (15)$$

and the decision function is

$$f(x) = \text{sign} \left(\sum_{i=1}^N \alpha_i^* y_i K(x_i, x_j) + b^* \right), \quad (16)$$

where α^* and b^* are the optimum solution of the problem.

The problem presented above is equivalent to solving a quadratic programming (QP) problem, which has a global optimum solution. Sequential Minimal Optimization (SMO) is considered a fast algorithm to train SVMs [36]. SMO breaks the QP problem into a series of smallest possible subproblems, which are then solved analytically. When all the Lagrange multipliers satisfy the Karush-Kuhn-Tucker (KKT) conditions within user-defined tolerance, the problem has been solved.

3.4. Classifier Evaluations. To evaluate the performance of the proposed malware detection model and the fitness-based adaptive weight strategy for PSO, a confusion matrix is calculated based on the output of the machine learning models, as tabulated in Table 1 (for a 2-class problem) [8]. True positive (TP) means the number of positive examples classified correctly, and the other measures are defined similarly.

There are many other evaluation metrics that measure the performance of learning models, such as *accuracy*, *recall*, *precision*, *F1*, ROC, and area under the ROC curve (AUC). Some of these are defined as follows:

$$\begin{aligned}
 \text{accuracy} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}, \\
 \text{recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\
 \text{precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\
 F1 &= \frac{(1 + \beta^2) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}} \\
 &= \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FN} + \text{FP}}
 \end{aligned} \quad (17)$$

$(\beta = 1)$.

Among these measures, *accuracy* represents the classifier's overall accuracy on the entire dataset; *precision* is the percentage of predicted benign applications that are truly benign; and *recall* is the percentage of correctly classified positive examples. In addition, *F1* integrates *recall* and *precision* into a measure of the overall effectiveness of the classification

model. When both *recall* and *precision* are high, the value of *F1* is high. The parameter β reflects the relative importance of *recall* and *precision*, and it is usually set to 1.0 [20]. The ROC curve describes the decision boundary for the relative costs of TP (*x*-axis) and FP (*y*-axis). The ideal point on the ROC curve is at the upper left corner, where all the positive examples are classified correctly. Meanwhile, the smoother the ROC curve is, the smaller the probability of overfitting is. The AUC is a complementary measure to the ROC curve. When the ROC curves intersect, a larger AUC indicates a better performance. Comparing the AUC between classifiers can indicate a dominance relationship [37].

4. Experiment and Discussion

In this section, we present the detailed methodology and results of the experiments. A 10-fold cross-validation is applied for the evaluations, which means that the classifiers were each executed 10 times to ensure that every portion of a split dataset was used as a training set [38]. We recorded the average values of the 10-fold cross-validation experiments as the final results for the subsequent comparisons. In addition, to eliminate the randomness in sample synthesis algorithms, we repeated the experiments 10 times and used the averaged results as the final results.

The experiment environment was an Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz with 32.0 GB of RAM running under a 64-bit Windows 7 operating system.

4.1. Dataset. The dataset contains both benign applications and malicious applications. As in most papers [5–8, 20], the benign applications in our study were downloaded from the Google Play store. We chose the most popular free applications in many categories. Malicious applications were collected from authoritative malicious application websites, such as <https://virusshare.com>, and covered most malware families, including Faker91, FakeInst, SkullKey, Aiplay, and DroidDream. Our collection aims to reflect the distribution of Android malware.

During an initial examination of the samples, we found that many malware files with the same size may be variants of other samples, but they have the same features. Therefore, we retained only one copy and removed others to reduce interference. In addition, we removed some samples whose sizes were too small because they may be scripted or light HTML applications. A total of 2125 applications were utilized, including 1109 benign applications and 1016 malware applications.

4.2. Experiments for Features. After decompiling, the permissions and functions of each application were extracted from the source code files. In total, 104 permission features were extracted. This number is less than the 136 possible permissions listed in the Manifest.permission group at API level 19 [39] because third-party applications are not allowed to use some permissions. We also extracted 14,233 functions by scanning .smali disassembler files. Finally, the total number of features, combining permissions and functions, was 14,337. Based on the features, the dataset with benign and

TABLE 2: Performance of different features used in SVM.

	Number	Accuracy	F1	Time (s)
Permissions	104	0.9264	0.9274	3.5253
Functions	14233	0.95620	0.9572	453.2207
Combined permissions and functions	14337	0.95621	0.9573	455.5641

malicious applications was represented as a binary matrix, in which a “0” denotes a feature that does not appear in the source code and a “1” denotes a feature that appears in the source code. Table 2 shows the performance of three kinds of features: permissions, functions, and combined permissions and functions. We use SVM as the classifier to compare their effects.

As shown in Table 2, the performance of the functions is approximately equal to the performance of the combined features that included both permissions and functions, which indicates that permissions as features, either alone or combined with functions, contribute little to malware detection. This occurs because the permissions can mainly be mapped to certain functions, while the functions contain all the information provided by the permissions. Therefore, the combined features contain redundant and noisy data. The larger the number of features, the greater the time cost of training and detection. Therefore, feature selection methods (e.g., IG) can be used to remove the noise features and reduce the quantity of features.

4.3. Experiments for Feature Selection. IG weights represent the importance of the features in the classification. By sorting the IG weights in descending order, the top features can be ranked and selected. The original dimensionality of the combined features was 14,337. This section describes how SVM was used to measure the accuracy of the top features ranked by the IG weights.

Figure 2 shows the accuracy of the SVM classifier using from 0.5% to 100% of the features weighted by IG. The curve indicates that the accuracy increases faster when fewer than 5% of the features are selected, whereas the accuracy increases more slowly when more than 5% of the features are selected. The accuracy achieved using 5% of the features improves very little in comparison with using all the features. Furthermore, according to the ROC and AUC of the selected features at different ratios, as shown in Figure 3, the AUC using 5% of the features is the largest. Therefore, we can conclude that the top 5% of the selected features are the best features; they achieve high accuracy and save computation time. This result is consistent with results from other studies [1, 7].

By ranking the IG weights in descending order, we obtain the top 20 features. The percentages of the selected features appearing in malicious versus benign applications are shown in Figure 4. Because of the differences in percentages, it is clear that these features can discriminate malicious applications from benign ones. Among these, the top 3 features used in malicious applications are SMS-related permissions, including SEND_SMS, RECEIVE_SMS, and READ_SMS.

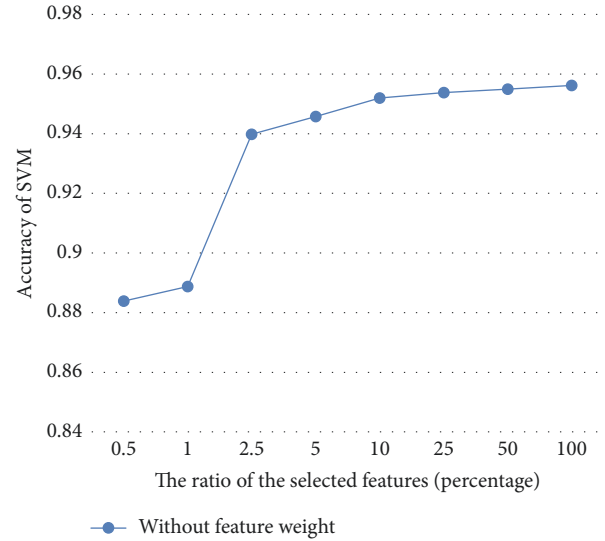


FIGURE 2: Accuracy of the features selected at different ratios by IG.

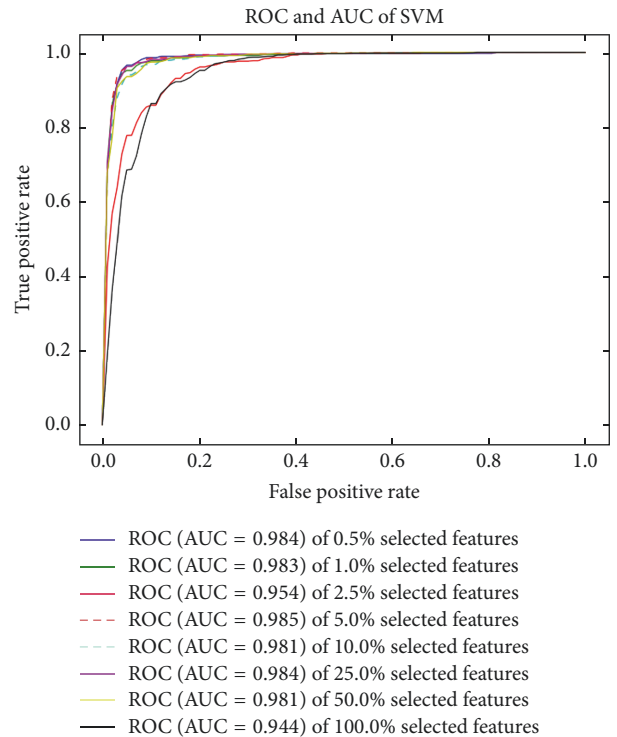


FIGURE 3: ROC and AUC of the features selected at different ratios by IG.

This is consistent with Android mobile-security-threat reports from Kaspersky, which indicate that SMS phishing and SMS Trojan viruses are the most widespread malicious attacks [40].

Table 3 shows the top 20 IG features with their feature weights. Among these, some features (such as Ljava/util/Set;->contains) look normal but are assigned relatively high weights. IG measures the amount of information in entropy that a feature brings to the classifier, depending

TABLE 3: Top 20 feature weights from the IG values.

Features	IG weights
android.permission.SEND_SMS	0.3426
android.permission.RECEIVE_SMS	0.3287
android.permission.READ_SMS	0.3158
Landroid/net/Uri;->getScheme	0.3112
Ljava/util/Set;->isEmpty	0.3082
Landroid/net/Uri;->getPath	0.3017
Landroid/view/ViewGroup;->removeAllViews	0.2974
Landroid/app/AlertDialog\$Builder;->setOnCancelListener	0.2960
Ljava/nio/CharBuffer;->flip	0.2935
Landroid/content/res/Resources;->getValue	0.2894
Landroid/graphics/Bitmap;->getPixel	0.2871
Ljava/util/Set;->contains	0.2867
Ljava/nio/CharBuffer;->allocate	0.2861
Ljava/util/Collections;->unmodifiableSet	0.2831
Landroid/net/Uri;->getQueryParameter	0.2819
Ljava/lang/Float;->floatToIntBits	0.2814
Ljava/lang/Double;->toString	0.2738
Ljava/lang/Readable;->read	0.2736
Landroid/net/Uri;->buildUpon	0.2696
Ljava/util/Collections;->singleton	0.2670

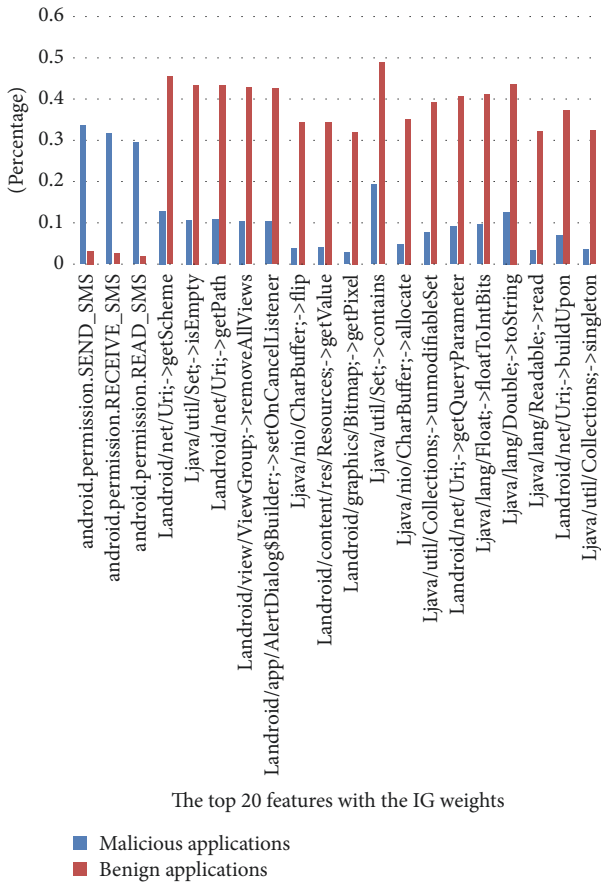


FIGURE 4: Percentages of malicious and benign applications using the top 20 features selected by IG.

on whether the feature is in or absent in the dataset [41]. The greater the percentage difference between the appearance and the absence of a feature is, the more the information it brings to the classification model and the greater the IG value is. On the contrary, a greater IG value means that the features are important for identifying the class of the examples. We conclude that the feature weights are assigned based on the actual usage of the features in the dataset and are not tempered by our experiences. However, the IG weights are small.

This situation may cause little change in the distribution of the feature space compared with the original feature space. Therefore, IG weights may result in only slight performance improvements in malware detection.

4.4. Performance Comparison with IG and PSO Feature Weights. PSO weights are initialized by a random matrix and adaptively updated to fit the highest accuracy achieved by the SVM detection model. In this part, both IG and PSO weights are compared to determine how much they improve the performance of malware detection. PSO with random inertia weight (RIW-PSO) is used to compute the PSO weights. Different ratios of features selected by IG are used in the SVM model.

Figure 5 shows three accuracy curves of different ratios of features selected both with and without the feature weights. The top two curves, which depict higher accuracy, show the results of using IG weights and PSO weights. They indicate that feature weights can improve accuracy of SVM detection model and that the accuracy is highest when using the PSO weights. Meanwhile, we find that the trends of the three curves are similar, regardless of whether feature weights are

TABLE 4: Top 20 feature weights by RIW-PSO with 5% selected features.

Features	PSO weights
Landroid/app/ActivityOptions;->makeScaleUpAnimation	0.9998
Landroid/graphics/Canvas;->drawColor	0.9984
android.permission.READ_EXTERNAL_STORAGE	0.9981
Landroid/util/Log;->v	0.9978
Landroid/net/NetworkInfo;->getType	0.9977
Landroid/content/DialogInterface;->cancel	0.9969
android.permission.CHANGE_WIFI_STATE	0.99611
android.permission.CALL_PHONE	0.99610
Landroid/graphics/Path;->close	0.9955
Landroid/view/WindowManager;->getDefaultDisplay	0.99399
Landroid/view/View;->isFocused	0.99397
android.permission.PROCESS_OUTGOING_CALLS	0.9936
Ljava/util/HashMap;->get	0.9932
Ljava/util/UUID;->getMostSignificantBits	0.9925
Ljava/util/LinkedList;->add	0.9923
Landroid/content/SharedPreferences\$Editor;->putLong	0.9918
Ljava/lang/StringBuilder;->toString	0.9916
android.permission.SET_WALLPAPER	0.9912
Landroid/content/Intent;->setType	0.9907
android.permission.WRITE_CONTACTS	0.9905

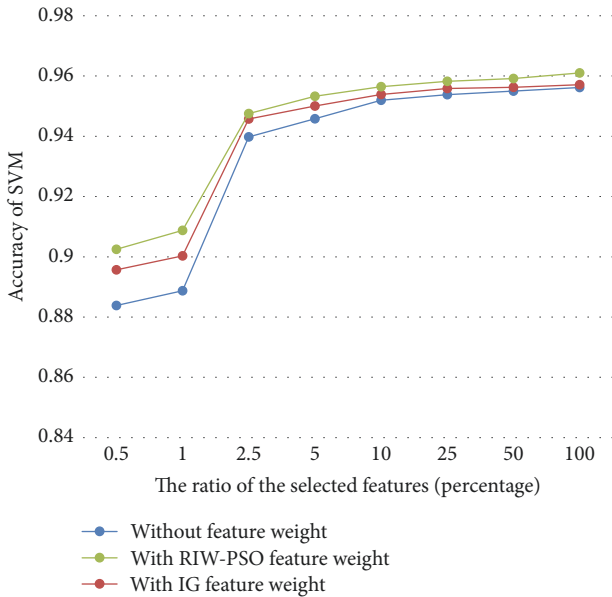


FIGURE 5: Accuracy of SVM with and without the feature weights.

used. It is possible that this occurs because the feature weights do not seriously affect the distribution of the feature space.

Table 4 shows the top 20 feature weights by PSO with 5% of the features selected at a certain time. Statistics from multiple experiments indicate that the PSO weights are not the same, mainly because PSO pursues the best overall fitness of the swarm and does not consider the state of each particle first. In our work, all the features are assigned the PSO

weights; they work together to ensure the best malware detection performance but follow the original rule.

Although the performance accuracies obtained using features with IG weights and PSO weights are only slightly different, as shown in Figure 5, there are large differences between the weights in IG and PSO in terms of which features are ranked highest and have higher weights (compare Tables 3 and 4). Meanwhile, the differences between the various percentages used in malicious applications and benign applications are clear, as shown in Figures 4 and 6. These top features with high weights have strong classification abilities. The IG weights are evaluated according to the importance of the features to the labels and without considering the correlations between the features, whereas the PSO weights are computed when weighted features work together to ensure the best malware detection performance. In addition, based on the different feature weights and top features, we can infer that the best feature weight values are not fixed; instead, they are assigned different values depending on different methods and the interactions between features.

4.5. Performance Comparison between Different Inertia Weight Strategies Adjusting PSO. Of the strategies introduced in Section 3.2, six inertia weight adjusting strategies for PSO are compared in this section: constant inertia weight (CIW) [29], random inertia weight (RIW) [30], linearly decreasing inertia weight (LDIW) [17, 30], chaotic decreasing inertia weight (CDIW) [31, 32], double exponential self-adaptive inertia weight (DESIW) [33], and fitness-based and chaotic adaptive inertia weight (FCAIW). First, these strategies are used to adjust the PSO algorithm; then, the improved PSO methods are wrapped into the SVM model on the Android

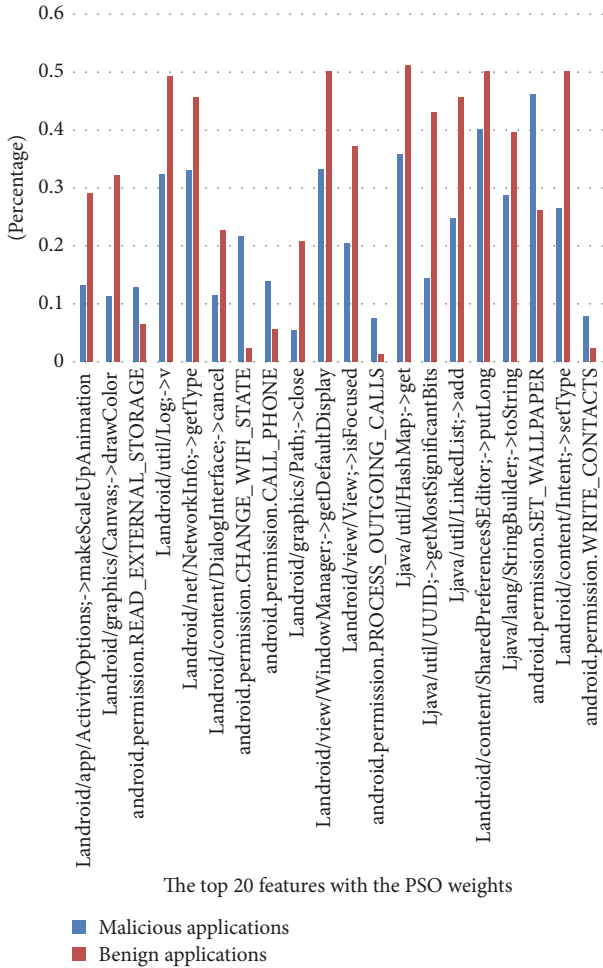


FIGURE 6: Percentages of malicious and benign applications using the top 20 features selected by RIW-PSO.

TABLE 5: The performance of six inertia weight strategies to adjust PSO for Android malware detection.

	CIW	RIW	LDIW	CDIW	DESIW	FCAIW
Accuracy	0.9520	0.9552	0.9517	0.9576	0.9511	0.9605
F1	0.9527	0.9563	0.9529	0.9588	0.9525	0.9616

dataset. 5% of the features selected by IG are used in this section. The accuracy and F1 of SVM are listed in Table 5.

The results of Figure 6 and Table 5 show that the PSO weights perform better than the IG weights; therefore, PSO works well for calculating adaptive weights for SVM to detect Android malware. The proposed FCAIW method performs considerably better than the other methods because it has a strong incentive to pursue the best fitness through the adaptive inertia weight adjustments and the addition of a chaotic sequence to maintain diversity. Consequently, FCAIW can find the optimal environmental changes. LDIW adjusts the inertia weight linearly within $[w_{min}, w_{max}]$, which is set to $[0.4, 1.4]$. When the inertia weight decreases gradually, the local search ability of the particles increases continuously. However, when the inertia weight requires more diversity,

the strategy can do nothing. However, CDIW, which is based on LDIW but involves an added chaotic term, can increase the diversity. Therefore, CDIW performs better than LDIW does. In CIW, the constant is set to 0.8 to maintain the overall search ability. However, CIW-PSO cannot adjust the inertia weight to balance the search well between global exploration and local exploitation. RIW uses a random value in the range $[0.4, 1.4]$ for the inertia weight to track the best fitness in a dynamic environment. It is difficult for RIW-PSO to yield suitable values when global exploration or local exploitation is required. However, after the number of iterations is large enough, the diversity can be increased. DESIW adjusts the inertia weight well at first. However, after several iterations, $R_i(t)$ becomes zero; therefore, all the particles' inertia weights become close to 1.0. At that point, DESIW is similar to RIW.

4.6. Comparison of Machine Learning Models with FCAIW-PSO Feature Weights. kNN, NB, DT, LR, SVM, AdaBoost, and k-means are popular machine learning methods for Android malware detection [20, 42, 43]. In this section, 5% of the features selected by IG are used. We first compute the IG and FCAIW-PSO weights based on the training dataset. Then, the IG weights and FCAIW-PSO weights are applied to the training dataset to train the machine learning models. Second, the IG and FCAIW-PSO weights are then applied to the testing dataset to generate mapped data, which are applied to the training models to determine the performance. Tables 6 and 7 show the accuracies of several machine learning models with and without feature weights.

We can obtain several conclusions from the results in Table 6: (1) SVM performs better than the other machine learning models; (2) the feature weights can improve the Android malware detection performance, except in the cases of the FCAIW-PSO weights with DT and the FCAIW-PSO/IG weights with AdaBoost; (3) the FCAIW-PSO weights perform better than the IG weights for the same feature space, with the exception of the FCAIW-PSO weights with DT; (4) the performance using only 5% of the selected features is close to that from using all the features. This result is consistent with the results in Section 4.3.

Feature weights can make similar samples compact and broaden the distribution of dissimilar samples; therefore, it is easy for kNN to classify dissimilar samples using Euclidean distance. NB depends on a multiplication of the posterior probability of each feature; therefore, the performance of high-dimensional features is worse than that of low-dimensional features. Moreover, NB adopts no measures to deal with the outliers, so its performance is worse than that of the other models. SVM not only uses the kernel function to map a low-dimensional space into a high-dimensional space to separate features easily, but also uses slack variables and penalty factor to deal with noisy data. Consequently, SVM has a high accuracy. Similarly, LR uses regularization to avoid overfitting with the training dataset and improve the testing ability. For DT, the results are shown in Table 7. During the training stage, the FCAIW-PSO weights are computed to ensure the best training performance. However, in the testing stage, the FCAIW-PSO weights reduce the testing performance compared to that of DT without weights in

TABLE 6: Comparison of the accuracies of machine learning methods with and without feature weights.

		kNN	NB	LR	SVM	DT	AdaBoost	<i>k</i> -means
Without feature weight	5% features	0.8844	0.7853	0.9398	0.9519	0.9300	0.9422	0.5047
	All features	0.9015	0.7707	0.9562	0.9562	0.9331	0.9586	0.5388
With IG feature weight	5% features	0.9094	0.7920	0.9495	0.9538	0.9307	0.9422	0.5415
	All features	0.9188	0.7865	0.9604	0.9570	0.9331	0.9586	0.5476
With FCAIW-PSO feature weight	5% features	0.9147	0.8432	0.9584	0.9605	0.9276	0.9422	0.6058
	All features	0.9235	0.8257	0.9635	0.9613	0.9288	0.9586	0.6152

TABLE 7: Performance of the DT model with and without feature weights.

		Training accuracy	Testing accuracy
With FCAIW-PSO feature weight	5% features	0.9993	0.9276
	All features	1.0	0.9288

Table 6. This result occurs mainly because the FCAIW-PSO weights cause overfitting during the testing stage. Therefore, it is necessary to take some measures to avoid overfitting. AdaBoost is an ensemble method that adaptively learns a series of weak classifiers to generate a boosted classifier by enhancing the weights of the misclassified examples. Based on the results in Table 6, we find that using any weights has no effect on AdaBoost’s performance; therefore, there is no need to compute the feature weights for AdaBoost to build models. *k*-means, as an unsupervised learning method, pays no attention to class labels, but the labels are valuable for classification purposes; therefore, *k*-means performs the worst. Based on this comparison, we recommend using supervised learning methods for malware detection.

5. Conclusion and Future Work

Android malware detection is a meaningful task that attracts thousands of researchers. One of the important methods for malware detection is to apply static analysis with machine learning algorithms. In this work, we also utilize static analysis to identify malware from benign applications. We apply feature weights based on IG and PSO methods to measure the importance of features for machine learning classification. More importantly, a fitness-based adaptive inertia weight strategy is proposed to improve the PSO algorithm. The purpose of the proposed strategy is to increase swarm diversity to achieve the best fitness (obtain the highest accuracy from the machine learning models). The results of experiments indicate that, in most cases, feature weights enhance the performance of Android malware detection and achieve better results than the same classifiers without weights. However, for models such as DT, feature weights cause overfitting with the training set, and the testing performance of these models decreases. Consequently, it is necessary to take some measures to avoid overfitting.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Acknowledgments

This paper is supported by the National Natural Science Foundation of China (nos. 61602052 and 61602489).

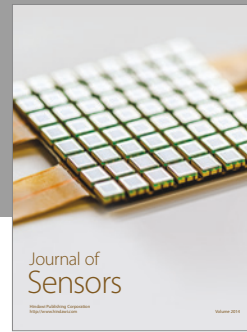
References

- [1] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, “A review on feature selection in mobile malware detection,” *Digital Investigation*, vol. 13, pp. 22–37, 2015.
- [2] S. Y. Yerima, S. Sezer, and G. McWilliams, “Analysis of bayesian classification-based approaches for android malware detection,” *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.
- [3] Y. Wang, J. Zheng, and C. Sun, “Quantitative security risk assessment of Android permissions and applications,” in *Data and Applications Security and Privacy XXVII*, pp. 226–241, Springer, New York, USA, 2013.
- [4] M. Zhang, Y. Duan, and H. Yin, “Semantics-aware android malware classification using weighted contextual api dependency graphs,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 1105–1116, Arizona, Ariz, USA, November, 2014.
- [5] N. Peiravian and X. Zhu, “Machine learning for Android malware detection using permission and api calls,” in *Proceedings of IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI ’13)*, pp. 300–305, IEEE, Herndon, VA, USA, November, 2013.
- [6] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, “Detecting android malicious apps and categorizing benign apps with ensemble of classifiers,” *Future Generation Computer Systems*, pp. 1–8, 2017.
- [7] D. Su, W. Wang, X. Wang et al., “Anomadroid: profiling Android application behaviors for identifying unknown malapps,” in *Proceedings of the 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom ’16)*, pp. 691–698, IEEE, Tianjin, China, August, 2016.
- [8] L. Cen, C. S. Gates, L. Si, and N. Li, “A probabilistic discriminative model for android malware detection with decompiled source code,” *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2015.
- [9] A. Shabtai, Y. Fledel, and Y. Elovici, “Automated static code analysis for classifying android applications using machine

- learning,” in *Proceedings of International Conference on Computational Intelligence and Security (CIS '10)*, pp. 329–333, IEEE, December, 2010.
- [10] W. Wang, Y. He, J. Liu, and S. Gombault, “Constructing important features from massive network traffic for lightweight intrusion detection,” *IET Information Security*, vol. 9, no. 6, pp. 374–379, 2015.
- [11] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, “Interpreting TF-IDF term weights as making relevance decisions,” *ACM Transactions on Information Systems*, vol. 26, no. 3, article no. 13, 2008.
- [12] Z. H. Deng, S. W. Tang, D. Q. Yang et al., “A comparative study on feature weight in text categorization,” in *Proceedings of the Asia-Pacific Web Conference*, pp. 588–597, Hangzhou, China, April, 2004.
- [13] N. Jankowski and K. Usowicz, “Analysis of feature weighting methods based on feature ranking methods for classification,” in *Proceedings of the International Conference on Neural Information Processing*, pp. 238–247, Shanghai, China, November, 2011.
- [14] S. Shankar and G. Karypis, “A feature weight adjustment algorithm for document categorization,” in *Proceedings of KDD-2000 Workshop on Text Mining*, Boston, USA, August, 2000.
- [15] C. Y. Tsai and C. C. Chiu, “Developing a feature weight self-adjustment mechanism for a K-means clustering algorithm,” *Computational Statistics & Data Analysis*, vol. 52, no. 10, pp. 4658–4672, 2008.
- [16] S. Lim, K. Lee, O. Byeon, and T. Kim, “Efficient iris recognition through improvement of feature vector and classifier,” *ETRI Journal*, vol. 23, no. 2, pp. 61–70, 2001.
- [17] A. Nickabadi, M. M. Ebadzadeh, and R. Safabakhsh, “A novel particle swarm optimization algorithm with adaptive inertia weight,” *Applied Soft Computing Journal*, vol. 11, no. 4, pp. 3658–3670, 2011.
- [18] M. Taherkhani and R. Safabakhsh, “A novel stability-based adaptive inertia weight for particle swarm optimization,” *Applied Soft Computing Journal*, vol. 38, pp. 281–295, 2016.
- [19] D. Wettschereck, D. W. Aha, and T. Mohri, “A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms,” *Artificial Intelligence Review*, vol. 11, no. 1, pp. 273–314, 1997.
- [20] Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in android,” in *Proceedings of International Conference on Security and Privacy in Communication Systems*, pp. 86–103, IEEE, Sydney, Australia, September, 2013.
- [21] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, “Exploring permission-induced risk in android applications for malicious application detection,” *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.
- [22] K. O. Elish, X. Shu, D. Yao, B. G. Ryder, and X. Jiang, “Profiling user-trigger dependence for android malware detection,” *Computers & Security*, vol. 49, pp. 255–273, 2015.
- [23] W. Zhang, T. Yoshida, and X. Tang, “A comparative study of TF*IDF, LSI and multi-words for text classification,” *Expert Systems with Applications*, vol. 38, no. 3, pp. 2758–2765, 2011.
- [24] W. Duch, T. Wiecek, J. Biesiada et al., “Comparison of feature ranking methods based on information entropy,” in *Proceedings of IEEE International Joint Conference on Neural Networks*, pp. 1415–1419, IEEE, July, 2004.
- [25] D. P. Vivencio, E. R. Hruschka Jr., M. Do Carmo Nicoletti, E. B. Dos Santos, and S. D. C. O. Galvão, “Feature-weighted k-nearest neighbor classifier,” in *Proceedings of 2007 IEEE Symposium on Foundations of Computational Intelligence (FOCI '07)*, pp. 481–485, April 2007.
- [26] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” *ICML*, vol. 97, pp. 412–420, 1997.
- [27] W. Wang, J. Liu, G. Pitsilis et al., “Abstracting massive data for lightweight intrusion detection in computer networks,” *Information Sciences*, pp. 1–14, 2016.
- [28] J. Kennedy, “Particle swarm optimization,” in *Encyclopedia of machine learning*, pp. 760–766, Springer, New York, USA, 2011.
- [29] Y. Li, Z.-H. Zhan, S. Lin, J. Zhang, and X. Luo, “Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems,” *Information Sciences*, vol. 293, pp. 370–382, 2015.
- [30] R. C. Eberhart and Y. Shi, “Tracking and optimizing dynamic systems with particle swarms,” in *Proceedings of the Congress on Evolutionary Computation*, vol. 1, pp. 94–100, Seoul, Republic of Korea, May 2001.
- [31] Y. Feng, G. F. Teng, A. X. Wang, and Y. M. Yao, “Chaotic inertia weight in particle swarm optimization,” in *Proceedings of the 2nd International Conference on Innovative Computing, Information and Control (ICICIC '07)*, 475 pages, IEEE, Kumamoto, Japan, September 2007.
- [32] M. Pluhacek, R. Senkerik, D. Davendra, Z. Kominkova Oplatkova, and I. Zelinka, “On the behavior and performance of chaos driven PSO algorithm with inertia weight,” *Computers and Mathematics with Applications*, vol. 66, no. 2, pp. 122–134, 2013.
- [33] P. Chauhan, K. Deep, and M. Pant, “Novel inertia weight strategies for particle swarm optimization,” *Memetic Computing*, vol. 5, no. 3, pp. 229–251, 2013.
- [34] S. Tong and D. Koller, “Support vector machine active learning with applications to text classification,” *The Journal of Machine Learning Research*, vol. 2, pp. 45–66, 2002.
- [35] S. R. Gunn, “Support vector machines for classification and regression,” ISIS Technical Report, University of Southampton, England, May, 1998.
- [36] J. C. Platt, “Using analytic QP and sparseness to speed training of support vector machines,” *Advances in Neural Information Processing Systems*, pp. 557–563, 1999.
- [37] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
- [38] “Cross-validation: evaluating estimator performance,” 2017, http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation.
- [39] android.Manifest.permission, 2017, <https://developer.android.com/reference/android/Manifest.permission.html>.
- [40] Kaspersky, “Android Mobile Security Threats,” 2016, <https://usa.kaspersky.com/internet-security-center/threats/mobile#.V2jse9Kqqko>.
- [41] G. Forman, “An extensive empirical study of feature selection metrics for text classification,” *Journal of Machine Learning Research*, vol. 3, pp. 1289–1305, 2003.
- [42] B. Sanz, I. Santos, C. Laorden et al., “Puma: permission usage to detect malware in android,” in *Proceedings of the International*

Joint Conference CISIS12-ICEUTE'12-SOCO'12 Special Sessions, pp. 289–298, Springer, Berlin, Germany, 2013.

- [43] H. Peng, C. Gates, B. Sarma et al., “Using probabilistic generative models for ranking risks of Android apps,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '12)*, pp. 241–252, ACM, Raleigh, NC, USA, October 2012.



Hindawi

Submit your manuscripts at
<https://www.hindawi.com>

