Feature Selection to Mine Joint Features from High-dimension Space for Android Malware Detection

Yanping Xu[1] *, Chunhua Wu[1], Kangfeng Zheng[1], Xinxin Niu[1] and Tianling Lu[2]
[1] School of Cyberspace Security, Beijing University of Posts and Telecommunications
Beijing, China, 100876
[e-mail: xyp_xyp@126.com]
[2] School of Information Technology and Network Security, People's Public Security University of China
Beijing, China, 100038
*Corresponding author: Yanping Xu

Received November 23, 2016; revised May 5, 2017; accepted June 3, 2017; published September 30, 2017

Abstract

Android is now the most popular smartphone platform and remains rapid growth. There are huge number of sensitive privacy information stored in Android devices. Kinds of methods have been proposed to detect Android malicious applications and protect the privacy information. In this work, we focus on extracting the fine-grained features to maximize the information of Android malware detection, and selecting the least joint features to minimize the number of features. Firstly, permissions and APIs, not only from Android permissions and SDK APIs but also from the developer-defined permissions and third-party library APIs, are extracted as features from the decompiled source codes. Secondly, feature selection methods, including information gain (IG), regularization and particle swarm optimization (PSO) algorithms, are used to analyze and utilize the correlation between the features to eliminate the redundant data, reduce the feature dimension and mine the useful joint features. Furthermore, regularization and PSO are integrated to create a new joint feature mining method. Experiment results show that the joint feature mining method can utilize the advantages of regularization and PSO, and ensure good performance and efficiency for Android malware detection.

Keywords: Android malware detection; feature selection; joint features; particle swarm optimization; regularization

1. Introduction

The increasing speed, power and storage space on Android mobile devices lead to more people using their Android infrastructures for online shopping, managing their finances, paying their bills and more places. Smart mobile devices run applications and store sensitive privacy information, such as contact lists, text messages, photos, geo-locations, users' accounts, etc. These sensitive privacy information is closely related to the interests of the users, which causes mobile devices to be a valuable target for cybercriminals to benefit from it. Meanwhile, the privacy data is vulnerable to leakage by malware attacks in various ways: accessing without permission, hijacking during transmission. Therefore, malware detection on Android devices is very meaningful.

To mitigate the threat on mobile devices, various efforts have been made to detect and analyze the Android malicious applications. Android malware detection methods are divided into three categories: static analysis, dynamic analysis and hybrid analysis. Static analysis approaches analyze the decompiled source codes without the execution of Android applications, which have been used by Enck et al. [1] and Felt et al.[2]. Dynamic detection methods analyze the application behaviors at runtime by monitoring behaviors indicative of Android application activity at runtime, which have been used in Crowdroid [3] and TaintDroid [4]. Hybrid analysis methods combine both static and dynamic techniques.

Machine learning methods are useful to detect Android malicious applications with many features, which are extracted from the Android applications. The bigger the number of the features is, the more information is contained for the machine learning classifiers. This motivates us to extract the features as fine as possible. Permissions and APIs in the source codes as features can provide much detailed detection information [5, 6]. In this work, the extracted features include not only the Android permissions and SDK APIs but also the user-defined permissions and third-party APIs. In the previous researches, user-defined permissions and third-party APIs are never analyzed for Android malware detection. If the feature granularity is too fine, the number of the extracted features from 1M size applications could be high to 20 thousand. Once the application size is high to dozens M, the number of the extracted features is up to hundreds of thousands. However, much data would cost much time and computation resource, and contain some redundant and irrelevant data. Therefore, feature selection methods can be adopted to select the least and best features for Android malware detection, and improve the quantity of the data set [7]. Moreover, the feature selection methods can mine the joint features from high-dimension feature space. These joint features contain most detection information, and they indicate that the training dataset are closely related to the testing dataset, so these joint features working together can identify the testing examples as far as possible [8, 9].

Feature selection is an important process to select the least and best feature subsets for building robust machine learning classifiers to detect Android malware in [8, 10]. In general, there are three kinds of feature selection methods: Filter, Wrapper and Embedded methods [10, 11]. Filter methods analyze the general characteristics of the features and calculate the relevance of a feature with respect to the class label without involving any machine learning algorithm, such as information gain (IG). Wrapper methods identify the best features based on the optimal performance of learning algorithms, such as particle swarm optimization (PSO). Embedded methods incorporate feature selection as a step of the machine learning process, such as decision tree (DT), L_1-support vector machine (SVM) and sparse logistic regression

(LR). Filter methods and any learning method are independent of each other, which lead to the structure of the classification model simple [11]. Wrapper methods, coupling with a specific learning model, often have good performance, but they cost expensive computational resources [12]. Embedded methods are usually more efficient, since they use the structure of the involved learning model to guide the features evaluation and searching. Currently, there are some embedded algorithms designed by the regularization methods, such as L_0-norm [13], L_1-norm [14] and $L_{1,2}$-norm[12]. These regularization methods select features with a projection matrix to encourage the row sparse, which are efficient and robust [9, 12]. In all, feature selection methods can remove the redundant and irrelevant features, so that the selected features can receive better detection performance than the original features.

After feature selection, machine learning methods, which have powerful data processing capabilities, are used to detect malware from the decomplicated sample set. Machine learning algorithms are trained to know the data with the training set. Then the obtained models are used to indentify the testing samples. It is suitable to utilize machine learning methods to detect Android malware. Popular machine learning methods include k-Nearest Neighbor (kNN), Naive Bayes(NB), LR, DT, SVM, Adaboost, k-Means, and more. They can identify the potentially malicious applications with high probability [15, 16]. This paper focuses on the feature detection based on machine learning methods. Firstly, we decompile APK files to extract a large number of permissions and APIs as features. Then three kinds of feature selection methods, including IG, PSO and $L_{2,1}$-norm regularization, are used to analyze the correlation between the features to mine the joint features. These joint features by PSO and $L_{2,1}$-norm regularization are relevant, and they work together to improve the performance of malware detection models. Moreover, both regularization and PSO can utilize the correlation between features to select features. Therefore, we integrate their advantage to create a new joint feature mining method. Our research makes significant contributions for Android malware static detection as follows:

- Combined features, not only from Android permissions and SDK APIs but also from the developer-defined permissions and third-party library APIs, are extracted from the Android decompiled source codes. The original dimensionality of the combined features is high to 28079, whose performance is better than that of the permissions and APIs respectively.
- PSO and regularization methods can mine and use the correlation between the features to select the least and best features, which provide the most detailed information for Android malware detection.
- Regularization and PSO methods are integrated to create a new joint feature mining method, which ensures good effectiveness and efficiency for Android malware static detection.

The rest is organized as follows: Section 2 reviews the related work on Android malware detection. Section 3 presents our research details on the methodology. Section 4 introduces and discusses the experiments to compare different methods. Section 5 concludes our research.

2. Related Work

Android malware detection techniques are similar to the techniques used on other OS platforms. They are fundamentally consisted of three main categories: static methods,

dynamic methods and hybrid methods [17]. Static methods analyze the source codes, binary and the API level without the execution of Android malware. Dynamic methods work through monitoring the execution of Android malware activity at runtime. Hybrid methods combine both the static and dynamic techniques. Static methods is based on the decompiled source codes. The application features are extracted from AndroidManifest.xml files, .java or .smali files, including permissions, functions, classes, data-flow and more.

Taking permissions as features, Enck et al. [1] proposed Kirin, which certificated an application with 9 predefined security rules at install time. The rules were to configure permissions and intent information to prevent the malicious applications installing. Felt et al. [3] built a tool called Stowaway, which detected over-privilege in decompiled Android applications based on the permission maps according to the relationship of the APIs and permissions. They found that about one-third applications were over-privileged. Sanz et al. [18] proposed PUMA, which extracted permissions from the applications and detects malwares through machine learning techniques including NB and Bayesian network. Wang et al. [16] explored the permission-induced risk in Android applications. They detected malwares based on the risky permissions with SVM and DT. The empirical results showed that the detection rate was 94.62% and false positive rate was 0.6%. Peng et al. [19] used the probabilistic generative models to calculate the risk scores based on the permissions requested by the Android applications. They found that simple NB algorithm was the promising risk scoring approach.

Taking APIs as features, Aafer et al. [15] developed a robust and light weight classifier named DroidAPIMiner, which extracted the critical API calls as the relevant features. They employed several machine learning classifiers including kNN, DT and SVM. The highest detection rate of kNN achieved 97.8%. Zhang et al. [20] proposed DroidSIFT to construct the dependence graphs with API calls. The graph databases were built for the known benign and malicious applications. They used the graph similarity queries to detect unknown applications based on the graph edit distance. The true positive rate of NB classifier was 93%. Arzt et al. [21] proposed a static taint-analysis system FlowDroid, which was context-, flow-, object-, filed-sensitive and lifecycle-aware taint analysis for Android applications. By handling the specific characteristics of Android, such as the lifecycle of an application or callback methods, FlowDroid conducted information flow analysis successfully. To address the challenge of implicit control flow transitions, Cao et al. [22] built EdgeMiner, which statically analyzed the Android framework. EdgeMiner generated the API summaries automatically that described the implicit control flow transitions. EdgeMiner could identify previously undetected leakage of the privacy sensitive data. Elish et al. [23] extracted the data-flow features on user trigger sensitive APIs and built TriggerMetric, which captured the data dependence relations between the user's actions and the sensitive operations. The values of TriggerMetric reflected the degree of sensitive operations that were triggered or intended by the users. The classification decision was made based on the values of TriggerMetric.

Taking multiple data as features, Peiravian et al. [6] proposed a framework, which extracted permissions and API calls as features, and used machine learning methods to detect Android malicious applications. The mentioned machine learning methods included SVM, DT and Bagging. The results showed that Bagging had the best performance for malware detection. Arp et al. [24] extracted the features based on requested permissions and API calls from AndroidManifest.xml and disassemble codes. Then they used SVM as the classifier. The detection rate of the malware samples achieved 94%. Wu et al. [25] implemented a system called DroidMat, which extracted the multiple features including permissions, components, intent messages passing and API calls. Then k-Means was applied to enhance the capability of

DroidMat. Lastly, kNN was used to classify the applications as benign or malicious applications. DroidMat had the better recall rate and high efficiency. Grace et al. [26] developed RiskRanker to analyze whether a particular application exhibited malicious behaviors based on control-flow and data-flow. It was helpful to analyze the encrypted native code and unsafe Dalvik codes. The results showed RiskRanker had the high efficacy and scalability to detect the zero-day malware. Yang et al. [27] proposed AppContext to construct a self-defined call graph based on the context of a security-sensitive behavior. Then AppContext detected the malware by classifying the security sensitive behaviors based on the extracted contexts.

Permissions and APIs are proved to be useful features to detect malware. So we also extract them from the decompiled files. Moreover, the extracted features include not only the Android permissions and Android SDK APIs but also the user-defined permissions and third-party APIs. In the previous research, user-defined permissions and third-party classes are never analyzed for malware detection. This is our first difference compared with other papers.

Feature selection is a crucial step in data processing. It chooses a best feature subset from the whole feature set based on the correlation between features and classes [28]. The selected features should contain the least features that have great impact on the performance of malware detection. Experiment results indicate that feature selection is useful for Android malware detection [5]. IG algorithm is widely used for feature selection based on the entropy difference [29]. [30] collected 2285 Android applications and extracted more than 9898 features. Then Chi-square (CHI), Fisher Score (FS), and IG methods were used to choose the top 50, 100, 200, 300, 500 and 800 features. Cen et al. [5] used IG and CHI for feature selection. They list the top 20 functions selected by IG and plotted a curve to show the performance of different ratio of the selected feature by IG and CHI. Experiment results show that IG and CHI were useful methods to select the best feature subset.

In our work, we mainly focus on the feature selection methods,including IG, PSO and $L_{2,1}$-norm regularization. They have been successfully applied to solve a large number of applications and difficult optimization problems [31-33]. [31] applied PSO to find the optimal feature subset, in which particle swarms found the best feature combinations when they flied within the subset space. [32] used PSO to accomplish multi-objective feature selection, whose goals were to maximize the classification performance and to minimize the number of features. [33] introduced a robust loss function, called Brownboost loss, which computed the feature quality and selectd the optimal feature subset to enhance robustness. [8] used $L_{1,2}$-norm on the projection matrix to achieve row-sparsity, which led to select the relevant features and learn the transformation simultaneously. Anyhow, feature selection is a meaningful data processing technology, which can minimize the classification error rate with the least number of features. So we use the feature selection to mine the joint features, and maximize the classification performance.

3. Methodology

The structure and process of Android malware detection based on the feature selection to mine joint features are depicted in **Fig. 1**, which consists of four major parts. The first one is reverse engineering, which decompiles APK files to the readable source code files, including AndroidManifest.xml and .smali files. The second part is feature extraction, which extracts features from the source code files. Then each application is represented as a single binary instance with permission and API features. Class label indicates whether the application is

benign or malicious. The third part is joint features mining, which analyzes the correlation between the features, mines the joint features from the high-dimension features set and moves the redundant information. The forth part is machine learning classifiers, which are trained and tested by the joint feature set and detects malware in the testing set.

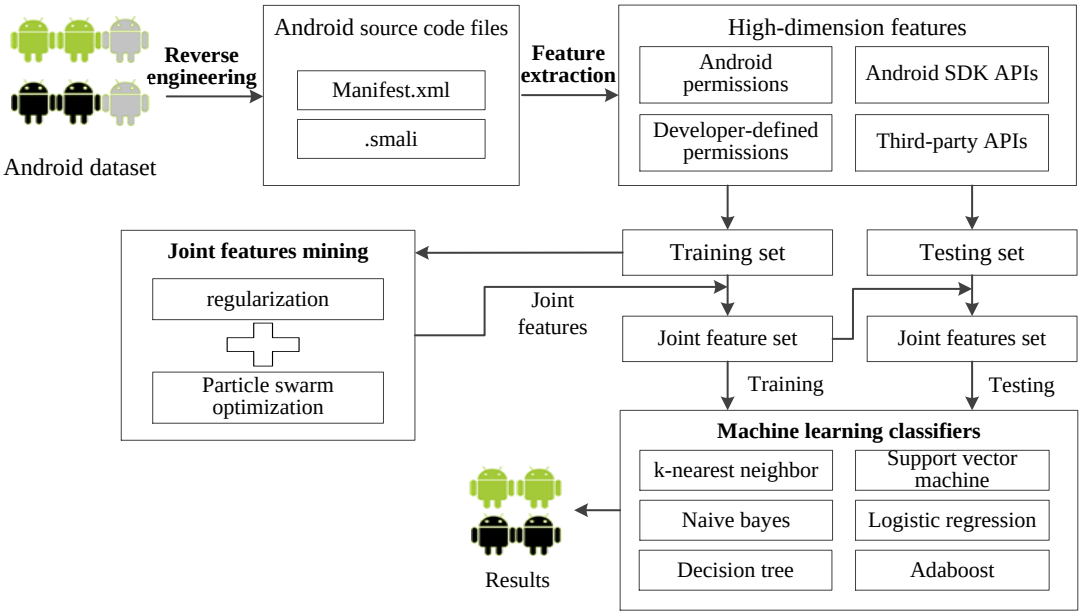

Fig. 1. The structure of Android malware detection based on the feature selection to mine the joint features

3.1 Reverse Engineering

Reverse engineering works to decompile APK files to obtain the source codes. Firstly, APK files are modified to .zip files. Then .zip files are unpacked to obtain AndroidManifest.xml and classes.dex. Secondly, we use java -jar AXMLPrinter2.jar AndroidManifest.xml >> AndroidManifest.txt, which converts the encoded AndroidManifest.xml to the readable AndroidManifest.txt. There are two methods to decompile .dex files. One is to compile .dex files to .jar files using dex2iar and jd-gui tool, and another one is to convert .dex files to .smali files using baksmali tool. Both of the decompiled files are readable and contain all information in different semantics. The .smali files are more readable in the basic rules, so we just adopt the second way. The bigger the application is, the more time reverse engineering costs. In the process of decompiling, the files are processed in batches, and about 1M data costs 2 seconds.

3.2 Feature Extraction

In Android platform, applications need to declare the permissions in AndroidManifest.xml when they call the sensitive functions, such as sendTextMessage(), mailto(), getAssets(), openConnection(). Android platform defines an API framework that consists of kinds of packages, classes and functions. Applications call these SDK APIs to interact with the underlying Android system. In addition, to satisfy the various needs of developers, developer-defined permissions (do not appear in the groups of Android system permissions, but are defined by developers themselves in the element <permission>) and the third-party APIs (are not defined by Google, but are called in Android applications) are always used.

Therefore, it is efficient to extract permissions and APIs as features to characterize and differentiate malware from benign applications [34].

In this work, we extract Android permissions and developer-defined permissions as features from "android.permission.XXX" in AndroidManifest.xml, and extract Android SDK APIs and third-party APIs as features from .smali files. The features extracted from each application are combined to obtain the union of the features. Then the Android application dataset is represented to be a binary matrix by the features union.

Cen et al. [5] have made experiments to indicate that permissions and APIs are complementary, and combined both permissions and APIs are helpful for detecting the malicious applications. However, if features extraction is much fine-grained, about 20 thousand features would be extracted from 1M application. Once the size of an application is high to dozens M, the number of the extracted features is high to a hundred thousand. The high-dimensional features can contain more detection information, but more time costs, and some repeated, meaningless and redundant features are contained. Therefore, feature selection methods can be used to reduce the dimensionality of features, avoid the curse of dimensionality and improve the detection performance.

3.3 Joint Features Mining

Machine learning methods are used for Android malware detection. The whole Android dataset is divided into the training dataset and the testing dataset by 10-fold cross-validation method **[5]**. Feature selection is working with the machine learning methods. In the training process, the features are selected by the feature selection methods from the training set. Then the selected features are used to simple the original training set, which is then input into the machine learning classifiers to train. In the testing process, the selected features are used to simple the original testing set, which is then input into the trained machine learning classifiers to identify the samples in the testing set.

In this paper, there are three kinds of feature selection methods, including IG, regularization and PSO, to mine the joint features. IG method assumes that all features are mutually independent, and selects features one by one through calculating the importance of the feature to the class, but it does not consider and utilize the correlation between the features to detect malware [10]. Regularization method applies the shared subspace learning to make high-dimension feature space project to a sparse shared subspace for exploiting the relational information of features [35]. PSO method works with the machine learning methods, and identifies the least and best features by fitting the highest accuracy of the training set [31]. Meanwhile, the highest accuracy depends on the correlation between the features.

3.3.1 Information Gain

Information Gain (IG) measures how much information a feature can bring into the classification system by knowing its presence or absence. It computes an IG value for every feature. The larger the IG value is, the more information the feature contains, and the more important the feature is [36].

Let x be a feature, whose category label is $y_k\,(k=0,1)$, each probability is $p(y_k)$, and the information entropy of a sample is defined as $H(y)$. Let $H(y\mid x)$ denote the conditional entropy of feature x, which represents information quantity x existing or not to the classification system. Therefore, the function for IG is:

$$IG(x) = H(y) - H(y \mid x)$$
$$= -\sum_{i=0}^{k} p(y_i)\log(p(y_i)) + p(x)\sum_{i=0}^{k} p(y_i \mid x)\log(p(y_i \mid x)) + p(\bar{x})\sum_{i=0}^{k} p(y_i \mid \bar{x})\log(p(y_i \mid \bar{x})) \quad (1)$$

IG method calculates IG value for each feature. IG values represent the importance of the features in the classification. By sorting the IG values in descending order, the top features can be ranked and selected.

3.3.2 Regularization methods

Regularization methods can be used to select features. In the original feature set, some features are unimportant to identify the sample, whose weights are set to 0 by the regularization methods. For example, the sample dataset is $X = (x_1, x_2, \cdots, x_N)$, the number of the samples is N. The lables of the dataset is $Y = (y_1, y_2, \cdots, y_N)$. Suppose there is a model $y_j = u_0 x_0 + u_1 x_1 + \cdots + u_n x_N + b$. Regularization methods make some $u_i (0 < i \le n) = 0$, if x_i is not irrelevant with $y_j (0 < j \le N)$. So regularization methods can make the model easy to understand and indicate which features are important to the labels. Meanwhile, regularization technology can keep the key features for the model to fit each sample [37]. L_1-norm and L_2-norm are common regularization methods. L_1-norm force the coefficient of the weak features to 0, so that the model becomes sparse. So L_1-norm is used as a feature selection method. L_2-norm can keep all features, but the coefficient of the weak features closes to 0. According to the coefficient value of the features, key features with high values can be selected [38]. In addition, $L_{2,1}$-norm regularization have been well studied and taken as an effective method for feature selection [39, 40]. In this paper, we use $L_{2,1}$-norm as the feature selection method.

Given training data set $X = (x_1, x_2, \cdots, x_N)$, the number of the samples is N, and the dimension of the features in n. and the associated class labels $Y = (y_1, y_2, \cdots, y_N)$. $L_{2,1}$-norm as a feature selection method aims to solve the following minimization problem.

$$\min_U \| U \|_{2,1} \; s.t. \quad XU = Y \quad (2)$$

$$\| u \|_{2,1} = \sum_{j=1}^{N} \sqrt{\sum_{i=1}^{n} u_{ij}^2} = \sum_{i=1}^{n} \| u_i \|_2 \quad (3)$$

where $\| \cdot \|_{2,1}$ is an $L_{2,1}$-norm, This optimization problem is solved with matrix method, which is implemented on the Lagrangian function to obtain the iterative solution.

The Lagrangian function of the problem in Eq. (2) is

$$L(U) = \| U \|_{2,1} - Tr(\Lambda^T (XU - Y)) \quad (4)$$

where $Tr(\cdot)$ is the matrix trace operator, and Λ is the Lagrange multipliers.

Taking the derivative of $L(U)$ with respect to U, and setting the derivative to zero, we have

$$\frac{\partial L(U)}{\partial (U)} = 2DU - X^T \Lambda = 0 \quad (5)$$

where D is a diagonal matrix with the i th diagonal element as

$$d_{ii} = \frac{1}{2 \| u_i \|_2} \tag{6}$$

Left multiplying the two sides of Eq. (5) by $X^T D^{-1}$, and using the constraint $X^T U = Y$, we obtain

$$2XU - XD^{-1}X^T \Lambda = 0 \tag{7}$$

$$2Y - XD^{-1}X^T \Lambda = 0 \tag{8}$$

$$\Lambda = 2(XD^{-1}X^T)^{-1}Y \tag{9}$$

Substitute Eq. (9) into Eq. (5), we have

$$U = D^{-1}X^T (XD^{-1}X^T)^{-1}Y \tag{10}$$

Since the problem in Eq. (2) is a convex problem, U is a global optimum solution to the problem if the Eq. (10) is satisfied. The convergence has been proven in [12]. In addition, D is dependent to U, which is also an unknown variable. We can use iterative approach to get U. Firstly, D is initialized to unit matrix. Then U is calculated with current D. The algorithm is over until the result converges.

3.3.3 Particle Swarm Optimization

Particle swarm optimization (PSO) is a evolutionary optimization method based on a populations (called a swarm). These particles are moving around in the search-space according to their personal best position (pbest) and global best positions (gbest) of the swarm. Improved positions are being discovered to guide the movements of the swarm [41]. PSO, as a feature selection method, can discover the best joint features when the particles search in the feature space. Meanwhile, the selected features can fit high detection accuracy of the training set [31]. The process is repeated until the optimal swarm positions are eventually discovered.

Let S be the number of particles. 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 $X_i = (x_{i1}, x_{i2}, \cdots, x_{in})$ and $V_i = (v_{i1}, v_{i2}, \cdots, v_{in})$, respectively. The process of the algorithm is as follows:

(1) Initialization: PSO is initialized with S particles, the number of the iteration is t, the number of the original features is n, the number of the selected features is $m(m \le n)$. At the first iteration $t = 0$, the position of the i th particle is $X_i(0) = (x_{i1}, x_{i2}, \cdots, x_{in})$. Among $x_{ij}(0 \le j \le n)$, m elements x_{ij} are randomly selected to set 1, and other elements x_{ij} are set to 0. The sequence number of x_{ij}, whose value is 1, is the sequence number of the selected features in the original feature set. The velocity of i th particle is $V_i(0) = (v_{i1}, v_{i2}, \cdots, v_{in})$, and the value of $v_{ij}(0 \le j \le n)$ is set to a distinct random value within the range [0,1].

(2) Building fitness: Fitness is used to measure the current results of the particles. In this work, the detection accuracy of machine learning methods with the selected features of the training set is used as the fitness of the particles.

(3) Finding optimal position: Comparing the current fitness with the best value in history, if the current fitness value is better than the best fitness value in history, current value is set to the new best one. The global optimal position is defined as

$p_g = (p_{g1}, p_{g2}, ..., p_{gd})$, which is updated according to the current value better than the history value, and the local optimal position is defined as $p_i = (p_{i1}, p_{i2}, ..., p_{id})$, which is updated according to the current value better than the history value.

(4) Updating position and velocity: During the search process, at iteration $t+1$, the position and velocity of the particles are updated according to formulas :

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + c_1 \cdot rand() \cdot (p_{ij}(t) - x_{ij}(t)) + c_2 \cdot rand() \cdot (p_{gj} - x_{ij}(t)) \quad (11)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \quad (12)$$

where w is the inertia weight, which balances the global searching ability and the local searching ability, c_1 and c_2 are two learning factors that control the social and cognitive components of the swarm, and $rand()$ consists of distinct random values within the range [0,1]. The inertia weight w is an important parameter that is set to a suitable value to strike a balance between global and local exploitation. Usually w is set to be a random value within the range [0.5, 1] to optimize the algorithm and $c_1 = c_2 = 2$.

In this work, Android application dataset is represented to be a binary matrix, so we use binary particle swarm optimization (BPSO) [42] to make the velocity v_{ij} of the particles discretize to 0 or 1 using Sigmoid.

$$s(v_{ij}) = \frac{1}{1+e^{-v_{ij}}} \quad (13)$$

$$v_{ij} = \begin{cases} 1, & if\ (rand() < s(v_{ij})) \\ 0, & if\ (rand() \geq s(v_{ij})) \end{cases} \quad (14)$$

In addition, the value of the position x_{ij} of the particles is 0 or 1, the max and min value of the velocity v_{ij} of each particle is within the range [-1,1].

$$v_{ij} = \begin{cases} 1.0, & if\ v_{ij} \geq 1.0 \\ -1.0, & if\ v_{ij} \leq -1.0 \end{cases}, \ j = 1, 2, \cdots, n \quad (15)$$

(5) Selecting fixed length m features: Taking the sequence numbers of the position x_{ij}, whose value is 1, are the sequence numbers of the selected features. If the length of the sequence number is m' $(m' < m)$, we randomly select $(m' - m)$ features from the original feature space to add the selected feature space. If the length of the sequence number is m' $(m' > m)$, we randomly select m features from the selected feature space. Anyhow, we must ensure the length of the selected features is m.

(6) Optimization termination condition: Iteration number is usually viewed as the termination condition of particles finding optimal positions. The value is set to 100.

3.4 Classifiers Evaluation

Comparing the true class label with predicted class label of testing set, confusion matrix [5] is generated based on the output of the machine learning methods that tabulated in a **Table 1**(for a 2 class problem): true positive (TP), true negative (TN), false positive (FP), and false

negative (FN). Of those, TP means the number of positive examples classified correctly, and the other measures are defined similarly.

Table 1. Confusion matrix

	Positive	**Negative**
True	true positive (TP)	true negative (TN)
False	false positive (FP)	false positive (FP)

There are other many evaluation metrics to measure the learning model, such as $accuracy$, $recall$, $precision$, $F1$, ROC and area under the ROC curve (AUC). They are defined as:

$$accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{16}$$

$$recall = \frac{TP}{TP + FN} \tag{17}$$

$$precision = \frac{TP}{TP + FP} \tag{18}$$

$$F1 = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall} = \frac{2 \times TP}{2 \times TP + FN + FP} (\beta = 1) \tag{19}$$

$accuracy$ is the overall accuracy of the whole datasets. $precision$ is the percentage of predicted positive applications actually being benign ones. $recall$ is the percentage of the correctly classified positive examples. In addition, $F1$ integrates $recall$ and $precision$ as a measure of the overall effectiveness of the classification models. When both $recall$ and $precision$ is high, the value of $F1$ is high. The parameter β reflects the relative importance of $recall$ and $precision$, and is usually set to 1.0 [15]. The ROC curve describes the decision boundary of 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 ROC curve is, the smaller the possibility ofover-fitting is. AUCis a complementary measure to ROC. When the ROC curves intersect, the bigger AUC indicates the better performance. Comparing the AUC between classifiers can indicate a dominance relationship [5].

4. Experimental Classification Results and Analysis

In this section, we present the detailed results of the experiments. 10-fold cross-validation is applied to split the dataset into 10 folds. Every fold is rotationally used as the testing set, and the other 9 folds are adopted as the training set. Finally, we recorded the average values of the 10-fold cross-validation experiments as the final results for the subsequent comparisons [43]. The advantages of 10-fold cross-validation are to void the over-fitting problem and not waste too much data, comparing a fixed arbitrary test set. Especially, 10-fold cross-validation is very suitable for the small dataset [44]. In addition, in PSO algorithm, the number of the particles $S = 10$, and the number of the iteration is 100. 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 is set as Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz with 32.0 GB RAM and 64 bit Windows 7 operating system.

4.1 Sample Dataset for Experiments

The dataset we use in the experiment contains benign applications and malicious applications. Benign applications were downloaded from the Google Play Store in March 2015, covering the top applications from 14 categories, which contained Office, Lifestyle, Travel, Children, Shopping, Education, Finance, Photography, Social, Tools, Reading, Multimedia, Sports, and Themes. We chose the most popular top free applications in each category. Malicious applications were collected from Tencent safety lab and http://sanddroid.xjtu.edu.cn:8080/#overview in August 2015, which covered many kinds of malware families, including Faker91, FakeInst, DroidDream, Geinimi, DroidKungFu2. Our collection aimed 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 might be scripted or light HTML applications. A total of 2125 applications were utilized, including 1109 benign applications and 1016 malware applications.

4.2 Feature Extracted from Decompiled Source Code

After decompiling, features were extracted from the source files, including Android permissions, Android SDK APIs, developer-defined permissions and third-party library APIs. In total, 429 permissions were extracted, including 104 Android permissions and 325 developer-defined permissions. After statistics, we found that the malicious applications clearly tended to request more SMS-related permissions, such as READ_SMS, WRITE_SMS, RECEIVE_SMS, and SEND_SMS. Moreover, we traversed all .smali files to collect 107084 APIs altogether, including 14233 Android SDK functions and 92851 third-party library functions. The granularity of extracted features was so fine that the number of features was very large. However, there was big difference between the samples in calling third-party library packages, so we just extracted the class-level features to be the third-party APIs, as shown in **Table 2**. At last, API features ware adopted, which contained 14233 Android SDK functions and 13417 third-party class-level features. Totally, 429 permissions and 27650 APIs were combined as the whole feature set for malware detection. Based on the features, the dataset with benign and malicious applications was represented as a binary matrix, in which a "0" denoted a feature that did not appear in the source code and a "1" denoted a feature that appeared in the source code. We use SVM as the classifier to compare the effects of the features.

Table 2. Feature extracted from decompiled source code

Features	Dimensions	Accuracy	$F1$	Time(s)
Android Permissions	104	0.9245	0.9258	0.0626
Developer-defined Permissions	325	-	-	-
Total Permissions	429	0.9349	0.9357	0.1388
Android SDK Functions	14233	0.9562	0.9574	6.2201
Third-party Classes	13417	-	-	-
Total APIs	27650	0.9563	0.9576	7.3490
Total Combined Permissions and APIs	28079	0.9811	0.9817	7.8943

As shown in **Table 2**, the higher the features dimension is, the higher the accuracy and $F1$ are, and the more time costs. The performance of the API features is better than that of the

permission features, but worse than that of the combined APIs and permissions. The accuracy of combined features is high to 98.11%. In addition, the results indicate that Android SDK functions contribute more accuracy to malware detection, while the third-party features, with a large number of data and costing more time, just contribute less than 1% to accuracy. Although total combined permissions and APIs contain more information, maybe there are still outliers and redundant data. Therefore, feature selection methods (e.g., IG) can be used to remove the noise features and reduce the quantity of features.

4.3 Comparison with feature selection methods

In this section, we use IG, regularization and PSO methods to select the best features through mining and using the correlation between the features. IG method is used to select features based on the descending order of the IG values of the features. Regularization method makes the weights of the unimportant features to 0 to select features. PSO selects features using the correlation between the features and ensures the best performance of the machine learning method. In this section, we will explore the performance of the three feature selection methods with the same number of the selected features. SVM is used as the machine learning classifier, and the whole feature set with 28079 combined permissions and APIs is used to select.

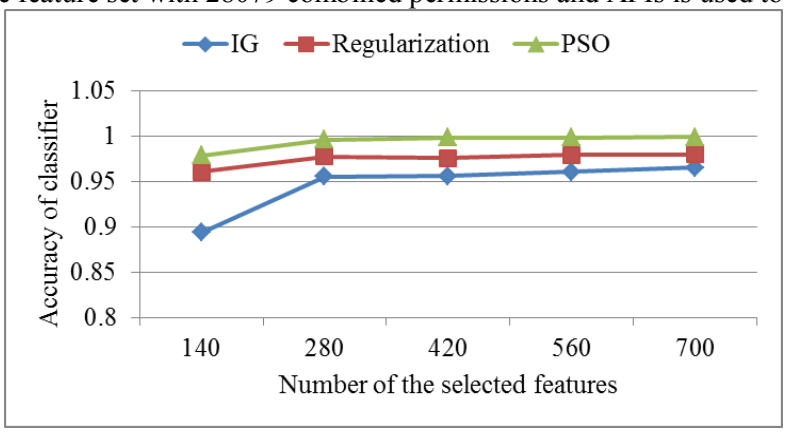

Fig. 2. Comparison with the feature selection methods

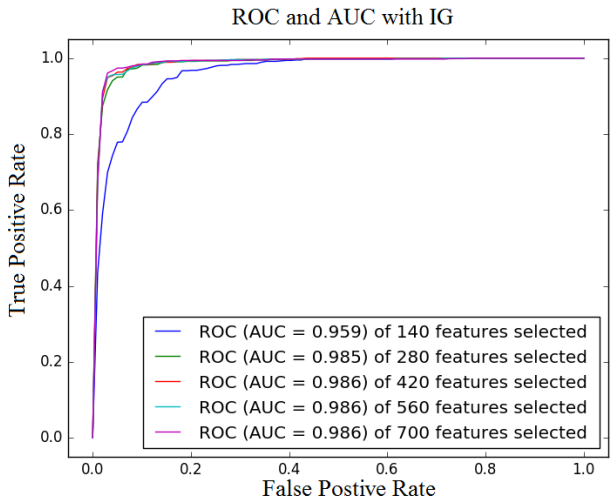

Fig. 3. ROC and AUC of the features selected by IG method

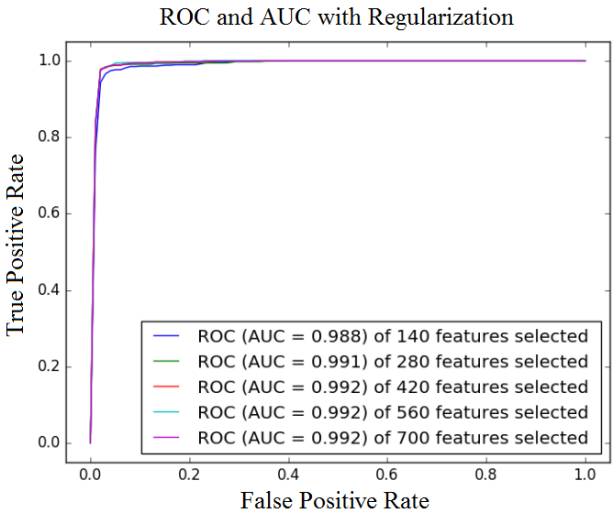

Fig. 4. ROC and AUC of the features selected by regularization method

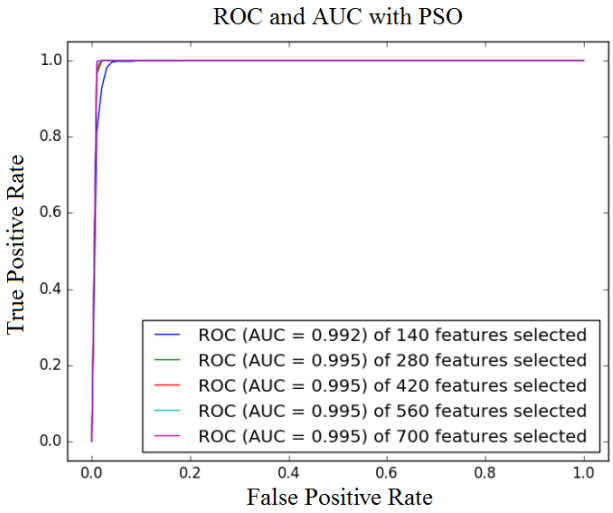

Fig. 5. ROC and AUC of the features selected by PSO method

In the experiment, we repeat the process of the feature selection methods many time, and use the feature selection methods to select a series of numbers of features from 100 to 1000. We record all the accuracy of the machine learning methods. Then we find that the law is very obvious when the number of the features selected are 140,280,420,560,700, so we just discuss these results in the manuscript.

Fig. 2-5 describe the performance of the SVM classification model with 140 to 700 features by the three feature selection methods mentioned above. **Fig. 2** shows three accuracy curves of different numbers of selected features. **Fig. 3-5** show ROC and AUC of the selected features using the three methods. As shown in these figures, we find PSO, whose accuracy and AUC are biggest and ROC is closest to the upper left, achieves the best performance, regularization comes second, and IG is the third. Even, the accuracy of more than 280 features with regularization and PSO methods is better than that of all features (97.51% in **Table 1**). Moreover, the curve in **Fig. 2** indicates that the accuracy increases faster when fewer than 280

features are selected, whereas the accuracy increases more slowly when more than 280 features are selected, and the performance of 280 features degrades very little compared to 700 features. Therefore, we can conclude that 280 is the suitable number of features, which can cover most information of the whole sample dataset, and ensure good effectiveness and efficiency. According to the accuracy of regulation and PSO higher than that of all features (97.51%), we conclude that not only the number of features determines the performance of the classifier, but also the interaction of joint features improves the performance. IG does not pay attention to the correlation between the features, but regularization and PSO methods mine and utilize the relevant features. So the performance of the features selected by regularization and PSO methods is better than that of IG method.

4.4 Top 20 joint features by different feature selection methods

In this section, IG, regularization and PSO methods are used to select the best features through mining and using the correlation between the features. The whole feature set with 28079 combined permissions and APIs is used to mine the joint features, because the whole feature set contains most information for the machine learning classifiers. The **Table 3** describes the top 20 joint features by different feature selection methods.

Table 3. The top 20 joint features by different feature selection methods

IG method	Regularization method	PSO method
android.permission.SEND_SMS	Landroid/telephony/SmsManager;->getDefault	Landroid/text/SpannableStringBuilder;->append
android.permission.RECEIVE_SMS	android.permission.SEND_SMS	Ljava/util/Queue;->clear
android.permission.READ_SMS	android.permission.SYSTEM_ALERT_WINDOW	Ljava/lang/Thread;->toString
Landroid/net/Uri;->getScheme	Landroid/telephony/SmsManager;->divideMessage	Ljava/lang/reflect/Field;->getAnnotation
Ljava/nio/CharBuffer;->flip	Landroid/content/Intent;->setDataAndType	Landroid/os/ParcelFileDescriptor;->writeToParcel
Ljava/util/Set;->isEmpty	Landroid/content/Context;->getApplicationInfo	Landroid/location/Location;->getAccuracy
Landroid/view/ViewGroup;->removeAllViews	Ljava/io/FileNotFoundException;->toString	Ljava/net/URL;->openStream
Landroid/app/AlertDialog;->setOnCancelListener	Ljava/util/LinkedHashSet;->add	Landroid/database/DataSetObserver;->onChanged
Ljava/nio/CharBuffer;->flip	Landroid/content/IntentFilter;->setPriority	Landroid/os/Parcel;->writeParcelable
Landroid/content/res/Resources;->getValue	Ljava/lang/Integer;->valueOf	Landroid/content/res/TypedArray;->peekValue
Landroid/graphics/Bitmap;->getPixel	Ljava/lang/CharSequence;->toString	Landroid/os/Binder;->onTransact
Ljava/util/Set;->contains	Ljava/io/OutputStream;->write	Landroid/app/Notification;->setFullScreenIntent
Ljava/lang/String;->toUpperCase	Ljava/lang/String;->split	Ljava/util/HashSet;->remove
Ljava/util/Collections;->unmodifiableSet	Landroid/os/SystemClock;->elapsedRealtime	Landroid/database/Cursor;->registerContentObserver
Landroid/net/Uri;->getQueryParameter	Ljava/lang/reflect/Method;->getParameterTypes	Ljava/lang/Thread;->interrupt

Ljava/lang/Float;->floatToIntBits	Landroid/content/Context;->startActivity	Landroid/content/IntentFilter;->addCategory
Landroid/widget/MediaController	Ljava/lang/Class;->getConstructor	Landroid/graphics/drawable/AnimationDrawable;->isOneShot
Ljava/lang/Double;->toString	Landroid/content/Intent;->getBooleanExtra	Landroid/os/Parcel;->createTypedArrayList
Ljava/lang/Readable;->read	Ljava/io/File;->createNewFile	Landroid/content/res/Resources;->getText
Landroid/net/Uri;->buildUpon	Ljava/lang/Thread;->getName	Landroid/view/VelocityTracker;->computeCurrentVelocity

As shown in **Table 3**, there is a big difference between the top 20 joint features by different methods. In addition, the selected joint features by PSO every time are different, because the process of feature selection is random. We just record one result in **Table 3**. It seems that the features with IG are from various packages and not related to each other. Meanwhile, the features with regularization method can reflect the similarity and relationship. For example, there are 3 SMS-related APIs and permissions, there are 5 APIs from android.content package, and there are 5 APIs from java.lang package. Similarly, the correlation between features can be found in the results with PSO method. There are several APIs appearing from android.content, and android.os packages. However, the top 20 features by three methods can reveal some sensitive behaviors of Android malware, such as SMS-related permissions and APIs containing android.permission.SEND_SMS, Landroid/telephony/SmsManager; ->getDefault, etc. The cause of the difference can be summarized that it is related to the principles of the methods. IG method assumes that all features are mutually independent. It selects features one by one through calculating the information entropy of the feature to the given class, and it does not consider the influence of the feature to the classifiers. Regularization method makes the weights of the unimportant features to 0 to select features. PSO method works with the classifiers, and identifies the best features by fitting the highest accuracy of the training set. Meanwhile, the highest accuracy depends on the joint action of the features.

4.5 Combination of the joint feature mining method

In section 4.4, the results indicate that PSO is a good feature selection method on high performance, but the iterative process costs a long time and high computations. Regularization and IG on performance are worse than PSO, but they win in high efficiency. Moreover, both regularization and PSO can utilize the correlation between features to select features. Therefore, we integrate their advantage to create a new joint feature mining method. In this section, we explore the performance of the joint feature mining method. SVM is used as the machine learning classifier, and the whole feature set with 28079 combined permissions and APIs is used to select.

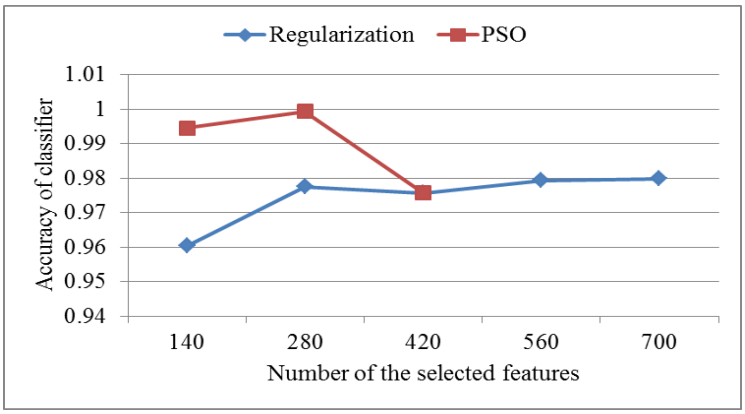

Fig. 6. The performance of the selected features by combined regularization and PSO methods

Fig. 6 shows the accuracy of the features selected by the joint feature mining method, which combines regularization and PSO methods. Firstly, 420 features are selected by regularization method from all features. Then 140 and 280 features are selected by PSO from the 420 features. Although the number of selected features by PSO is less than that of regularization method, the accuracy of PSO is higher than that of regularization. The result indicates that the joint feature mining method can improve more performance, comparing the separate regularization and PSO method, as shown in **Fig. 2**. That is because regularization and PSO not only eliminate the outliers but also make full use of the correlation of the selected features, which providers more information to identify samples. Meanwhile, the joint feature mining method reduces the amount of computation, comparing PSO selecting features from all the features. In sum, the joint feature mining method can achieve high effectiveness to make up the loss of efficiency.

4.6 Machine learning classification models comparison with feature selection scheme

In above sections, SVM is proved to receive high performance for Android malware detection. Actually, kNN, NB, DT, LR, SVM, Adaboost, k-Means are popular machine learning methods for Android malware detection [15,18,19]. In this section, we explore the performance of these machine learning methods, and the joint feature mining method combined regularization and PSO methods is used to improve the performance.

Table 4. Machine learning classification models comparison with selected features

		NB	kNN	LR	SVM	DT	Adaboost	k-Means
140 features by PSO	Accuracy	0.9163	0.9130	0.9970	0.9988	0.9994	0.9994	0.5436
	*F*1	0.8281	0.9156	0.9970	0.9988	0.9994	0.9994	0.5305
280 features by PSO	Accuracy	0.8200	0.9209	0.9994	1.000	1.000	1.000	0.5238
	*F*1	0.8303	0.9239	0.9994	1.000	1.000	1.000	0.5177
420 features by regularization	Accuracy	0.8273	0.9355	0.9757	0.9769	0.9197	0.9642	0.4500
	*F*1	0.8408	0.9376	0.9761	0.9773	0.9207	0.9650	0.3804
All features	Accuracy	0.7665	0.8765	0.9751	0.9811	1.000	1.000	0.6187
	*F*1	0.7623	0.8742	0.9757	0.9819	1.000	1.000	0.6343

Table 4 shows the performance of different machine learning classifiers with the selected features, which are selected by the joint feature mining method. Firstly, comparing between

different joint features, the performance of features selected by PSO is the better than that without PSO, because these features selected by PSO contain most information for malware detection. In addition, For NB, kNN and LR, the performance of all features is worse than that of 420 features selected by regularization. On the contrary, for the other classifiers, the performance of all features is better than that of 420 features selected by regularization. But more features must cost more time and computation resource. So the joint feature mining method is proved again to be helpful to improve performance of malware detection.

Secondly, comparing different machine learning classification models, the differences of the performance are obvious. DT and Adaboost are equally good, and better than other classifiers, because Adaboost is an ensemble method, which is adaptive to learn a series of weak classifiers to generate a boosted one through enhancing the weight of the misclassified example, so that the method can obtain the best performance. DT can be viewed as a kind of embed feature selection method, where the feature selection process works as a step of the learning process, and pruning algorithms are used to avoid over-fitting problem, so DT can reach good performance. SVM not only uses kernel function to map low-dimension space into high-dimension space to separate features easily, but also uses the slack variables and penalty factor to deal with noise data, so SVM has high accuracy. LR uses regularization to avoid over-fitting and ensures high accuracy in testing set. kNN relays on the instances with nearest neighbors, which has no solution to outliers, so the performance is not good, and when the size of the dataset is big, it would cost more time to calculate nearest neighbors. NB depends on the multiplication of posterior probability of each feature, and neither takes measures to deal with the outliers. K-Mean, as an unsupervised learning method, does not pay attention to class labels, which are valuable for classification. Comparatively, supervised learning methods are recommended for malware detection.

5. Conclusion

With the development of mobile devices, Android security attracts much attention, and Android malware detection is regarded as an important research task. To obtain most detailed information from the Android applications, permissions and APIs, not only from Android permissions and SDK APIs but also from developer-defined permissions and third-party library APIs, are extracted as features from the decompiled source codes. However, these fine-grained features contain redundant data, so feature selection methods, such as IG, regularization and PSO, are used to mine the useful joint features for classification and reduce the feature dimension. Experiment results indicate that there is a big difference between the joint features by the three methods. IG calculates the importance of the features based on assuming that all features are mutually independent. This means that IG takes no consideration of the combined action of the selected features. While regularization and PSO method can mine and use the correlation between the features, which provides more information for machine learning methods. In addition, regularization and PSO are integrated to create a new joint feature mining method. Experiment results show that the joint feature mining method can utilize the advantages of regularization and PSO, ensure good performance and efficiency for Android malware detection.

Acknowledgments

This paper is supported by the National Nature Science Foundation of China (No. 61602052, NO. 61602489).

References

[1] W. Enck, M. Ongtang, P. McDaniel, "On lightweight mobile phone application certification," in *Proc. of the 16th ACM conference on Computer and communications security*, ACM, pp.235-245, November 9-13, 2009. Article (CrossRef Link).

[2] W. Enck, P. Gilbert, S. Han, et al, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol.32, no.2, pp.5-21, March, 2014. Article (CrossRef Link).

[3] A. P. Felt, E. Chin, S. Hanna, et al, "Android permissions demystified," in *Proc. of the 18th ACM conference on Computer and communications security*, pp.627-638, October 17-21, 2011. Article (CrossRef Link).

[4] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proc. of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp.15-26, October 17-21, 2011. Article (CrossRef Link).

[5] L. Cen, C. S. Gates, L. Si, et al, "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, July-Augest. 1, 2015. Article (CrossRef Link).

[6] N. Peiravian, X. Zhu, "Machine learning for Android malware detection using permission and api calls," in *Proc. of 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp.300-305, November 7-9, 2013. Article (CrossRef Link).

[7] J. Crussell, C. Gibler, H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in *Proc. of Computer Security–ESORICS 2012*, pp.37-54, September 10-12, 2012. Article (CrossRef Link).

[8] Q. Gu, Z. Li, J. Han, "Joint feature selection and subspace learning," in *Proc. of IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, pp. 1294-1300, July 16-22, 2011. Article (CrossRef Link).

[9] J. Wen, Z. Lai, Y. Zhan, et al, "The $L_{1,2}$ -norm-based unsupervised optimal feature selection with applications to action recognition," *Pattern Recognition*, vol.60, pp. 515-530, June, 2016. Article (CrossRef Link).

[10] M. A. Hall, "Correlation-based feature selection for machine learning," *The University of Waikato*, April, 1999. Article (CrossRef Link).

[11] H. Liu, H. Motoda, R. Setiono, et al, "Feature selection: An ever evolving frontier in data mining," in *Proc. of FSDM 2010 : International Workshop on Feature Selection in Data Mining*, pp.4-13, January 20-21, 2010. Article (CrossRef Link).

[12] F. Nie, H. Huang, X. Cai, et al, "Efficient and robust feature selection via joint $L_{1,2}$ -norms minimization," in *Proc. of Advances in neural information processing systems*, pp.1813-1821, December 6-9, 2010. Article (CrossRef Link)

[13] K. Huang, I. King, M. R. Lyu, "Direct zero-norm optimization for feature selection," in *Proc. of ICDM workshops 2008. Eight IEEE international conference on data mining workshops*, pp.845-850, December 15-19, 2008. Article (CrossRef Link).

[14] A. Y. Yang, S. S. Sastry, A. Ganesh, et al, "Fast L_1 -minimization algorithms and an application in robust face recognition: A review," in *Proc. of 2010 IEEE International Conference on Image Processing*, pp.1849-1852, May 23, 2010. Article (CrossRef Link).

[15] Y. Aafer, W. Du, H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Proc. of International Conference on Security and Privacy in Communication Systems*, pp.86-103, September 25-28, 2013. Article (CrossRef Link).

[16] Y. Wang, J. Zheng, C. Sun, et al. "Quantitative security risk assessment of Android permissions and applications," in *Proc. of Data and Applications Security and Privacy XXVII*, pp.226-241, July 15–17, 2013. Article (CrossRef Link).

[17] F. Ali, B. A. Nor, S. Rosli, W. A. W. Ainuddin, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol.13, pp.22-37, March, 2015. Article (CrossRef Link).

[18] B. Sanz, I. Santos, C. Laorden, et al. "Puma: Permission usage to detect malware in Android," in *Proc. of International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pp.289-298, September, 2013. Article (CrossRef Link).
[19] H. Peng, C. Gates, B. Sarma, et al, "Using probabilistic generative models for ranking risks of Android apps," in *Proc. of the 2012 ACM conference on Computer and communications security*, pp.241-252, October 16-18, 2012. Article (CrossRef Link).
[20] M. Zhang, Y. Duan, H. Yin, et al, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp.1105-1116, November 3-7, 2014. Article (CrossRef Link).
[21] S. Arzt, S. Rasthofer, C. Fritz, et al, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM SIGPLAN Notices*, vol.49, no.6, pp.259-269, June, 2014. Article (CrossRef Link).
[22] Y. Cao, Y. Fratantonio, A. Bianchi, et al, "EdgeMiner: Automatically detecting implicit control flow transitions through the Android framework," in *Proc. of 2015 Network and Distributed System Security (NDSS) Symposium*, February 8-11, 2015. Article (CrossRef Link).
[23] K. O. Elish, X. Shu, D. D. Yao, et al, "Profiling user-trigger dependence for Android malware detection," *Computers & Security*, vol.49, pp.255-273, November, 2015. Article (CrossRef Link).
[24] D. Arp, M. Spreitzenbarth, M. Hubner, et al, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. of 2014 Network and Distributed System Security (NDSS) Symposium*, February 23-26, 2014. Article (CrossRef Link).
[25] D. J. Wu, C. H. Mao, T. E. Wei, et al, "Droidmat: Android malware detection through manifest and api calls tracing," in *Proc. of 2012 Seventh Asia Joint Conference on Information Security (Asia JCIS)*, pp.62-69, August 09-10, 2012. Article (CrossRef Link).
[26] M. Grace, Y. Zhou, Q. Zhang, et al, "Riskranker: scalable and accurate zero-day Android malware detection," in *Proc. of the 10th international conference on Mobile systems, applications, and services*, pp.281-294, June 25-29, 2012. Article (CrossRef Link).
[27] W. Yang, X. Xiao, B. Andow, et al, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proc. of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, pp.303-313, May 16-24, 2015. Article (CrossRef Link).
[28] J. Krawczuk, T. Łukaszuk, "The feature selection bias problem in relation to high-dimensional gene data," *Artificial intelligence in medicine*, vol.66, pp.63-71, November, 2016. Article (CrossRef Link).
[29] H. S. Ham, M. J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in *Proc. of 2013 International Conference on ICT Convergence (ICTC)*, pp.490-495, October 14-16, 2013. Article (CrossRef Link).
[30] A. Shabtai, Y. Fledel, Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Proc. of 2010 International Conference on Computational Intelligence and Security (CIS)*, pp.329-333, January, 2010. Article (CrossRef Link).
[31] X. Wang, J. Yang, X. Teng, et al, "Feature selection based on rough sets and particle swarm optimization," *Pattern Recognition Letters*, vol. 28, no.4, pp.459-471, November, 2007. Article (CrossRef Link).
[32] B. Xue, M. Zhang, W. N. Browne, "Particle swarm optimization for feature selection in classification: a multi-objective approach," *IEEE transactions on cybernetics*, vol. 43, no.6, pp.1656-1671, December, 2013. Article (CrossRef Link).
[33] P. Wei, Q. Hu, P. Ma, et al. "Robust feature selection based on regularized brownboost loss," *Knowledge-Based Systems*, vol.54, pp.180-198, September, 2013. Article (CrossRef Link).
[34] S. H. Seo, A. Gupta, A. M. Sallam, et al, "Detecting mobile malware threats to homeland security through static analysis," *Journal of Network and Computer Applications*, vol.38, pp.43-53, June, 2014. Article (CrossRef Link).

[35] R. He, T. Tan, L. Wang, et al., "$L_{1,2}$ regularized correntropy for robust feature selection," in *Proc. of 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.2504-2511, June 16-21, 2012. Article (CrossRef Link).

[36] Y. Yang, J. O. Pedersen, "A comparative study on feature selection in text categorization," in *Proc. of the Fourteenth International Conference on Machine Learning (ICML 1997)*, pp.412-420, July 8-12, 1997. Article (CrossRef Link).

[37] Y. Yang, H. T. Shen, Z. Ma, et al, "$L_{1,2}$ -norm regularized discriminative feature selection for unsupervised learning," in *Proc. of IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, pp.1589-1595, July 16-22, 2011. Article (CrossRef Link).

[38] A. Y. Ng, "Feature selection L_1 vs. L_2 regularization, and rotational invariance," in *Proc. of the twenty-first international conference on Machine learning*, pp.78-86, July 4-8, 2004. Article (CrossRef Link).

[39] J. Wen, Z. Lai, Y. Zhan, et al, "The $L_{2,1}$ -norm-based unsupervised optimal feature selection with applications to action recognition," *Pattern Recognition*, vol.60, pp.515-530, June, 2016. Article (CrossRef Link).

[40] C. X. Ren, D. Q. Dai, H. Yan, "Robust classification using $L_{2,1}$ -norm based regression model," *Pattern Recognition*, vol.45, no.7, pp.2708-2718, January, 2012. Article (CrossRef Link).

[41] J. Kennedy, "Particle swarm optimization," *Encyclopedia of machine learning*, pp.760-766, May, 2011. Article (CrossRef Link).

[42] R. C. Eberhart, J. Kennedy, "A discrete binary version of the particle swarm algorithm," in *Proc. of the IEEE Conference on Systems, Man and Cybernetics*, pp.4104-4109, September 3-5, 1997. Article (CrossRef Link).

[43] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. of the Fourteenth International Joint Conference on Artificial Intelligence*, pp.1137-1145, August 20-25, 1995. Article (CrossRef Link).

[44] Cross validation. http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation, August, 2013.

Yanping Xu is a Ph.D. student in computer science in Beijing University of Posts and Telecommunications. She is currently a member of Information Security Center. She received her MS degree in Beijing University of Posts and Telecommunications in 2013 majoring in electronics and communication engineering. Her research interests include Android malware detection and privacy protection, network and information security.

Chunhua Wu is a teacher in the Information Security Center, School of Cyberspace Security, Beijing University of Posts and Telecommunications. She received her Ph.D degree in Beijing University of Posts and Telecommunications in 2008. Her research interests include network and information security, applied more than seven important projects.

Kangfeng Zheng is a professor in the Information Security Center, School of Cyberspace Security, Beijing University of Posts and Telecommunications. He received his Ph.D degree in Beijing University of Posts and Telecommunications in 2006. His research interests include networking and system security, network information processing and network coding. He has published over 50 technical papers in international conferences and journals. He has presided over a number of national scientific research projects and received a number of national and provincial awards.

Xinxin Niu is a professor and director in the Information Security Center, School of Cyberspace Security, Beijing University of Posts and Telecommunications. She received her Ph.D degree in the Chinese University of Hong Kong. Her research interests include information security, information hiding and digital watermarking, digital content and its security. She has presided over a number of national scientific research projects and received a number of national and provincial awards.

Tianling Lu is a teacher in the Information Security Center, School of Information Technology and Network Security, People's Public Security University of China. He received his Ph.D degree in Beijing University of Posts and Telecommunications in 2013. His research interests include networking and system security, network information processing and network coding.