15 An Evaluation of One-Class Feature Selection and Classification for Zero-Day Android Malware Detection

Yang Wang and Jun Zheng

Abstract

Security has become a serious problem for Android system as the number of Android malware increases rapidly. A great amount of effort has been devoted to protect Android devices against the threats of malware. Majority of the existing work use two-class classification methods which suffer the overfitting problem due to the lack of malicious samples. This will result in poor performance of detecting zero-day malware attacks. In this paper, we evaluated the performance of various one-class feature selection and classification methods for zero-day Android malware detection. Unlike two-class methods, one-class methods only use benign samples to build the detection model which overcomes the overfitting problem. Our results demonstrate the capability of the one-class methods over the two-class methods in detecting zero-day Android malware attacks.

Keywords

One-class classification · One-class feature selection · Android malware · Malware detection · Performance evaluation

This paper is part of Yang Wang's dissertation which has not been published in other conference or journal.

Y. Wang
Ultramain Systems, Inc., Albuquerque, NM, USA

J. Zheng (✉)
Department of Computer Science and Engineering, New Mexico Institute of Mining and Technology, Socorro, NM, USA
e-mail: jun.zheng@nmt.edu

15.1 Introduction

Android is the most popular operating system for mobile devices, which has a share of around 87.0% of the global smartphone market in 2019 (https://www.idc.com/promo/smartphone-market-share/os). The official Android application (app) market, Google Play, hosted 2.8 million apps in Sept. 2019. On the other side, Android is also the primary target of mobile malware. According to the Threat Intelligence Report of Nokia, smartphones infected with Android malware nearly doubled in 2016 (https://securityledger.com/2017/03/android-malware-doubled-in-2016-adding-to-mobile-malware-problem/). A great amount of effort has been devoted to protect Android devices against the threats of malware [1–4]. Most of those work require known Android malware for applying two-class classification methods. However, collecting a set of representative samples covering various malicious behaviors is very hard if not impossible. Another problem of using two-class classification is the issue of class imbalance [5]. The classification performance may deteriorate due to the significant differences of class prior probabilities. Generally, there are far more benign samples than malicious samples in the training set which causes the overfitting of trained models. These problems will result in poor detection performance, especially for zero-day Android malware.

One-class classification (OCC), sometimes referred as anomaly or novelty detection, learns and builds the classification model from samples of a single class, i.e. the target class [6]. A new sample will be classified by the trained model as target class or unknown (outlier) class. For Android malware detection, the classification model is learned only with benign samples, which overcomes the problems of collecting representative malicious samples and class imbalance

S. Latifi (ed.), *17th International Conference on Information Technology–New Generations (ITNG 2020)*, Advances in Intelligent Systems and Computing 1134,
https://doi.org/10.1007/978-3-030-43020-7_15

of using the two-class classification method. On the other hand, feature selection, which is applied before classification to remove redundant or misleading information and reduce computational complexity, is not easy for one-class classification problem. Many methods used for two-class/multi-class feature selection are not viable for one-class feature selection since they need samples from multiple classes. In this paper, we performed an evaluation of various one-class feature selection and classification methods for identifying zero-day Android malware. We demonstrated that one-class methods achieve better performance than two-class methods in detecting unknown Android malware.

15.2 One-Class Feature Selection and Classification for Zero-Day Android Malware Detection

15.2.1 Overview

Figure 15.1 shows the workflow of using one-class feature selection and classification for zero-day Android malware detection. In the training stage, the training dataset is prepared with samples from the target class, i.e. benign apps. Through feature extraction, each sample is transformed to a vector formed by various features such as requested permissions, API calls etc. To reduce the computational complexity, a one-class feature selection algorithm is applied to remove those irrelevant and redundant features. Finally, a one-class classification model is built with the selected features. During the predication stage, a new sample is transformed to a vector of selected features. Then the trained OCC model is used to classify the sample as benign or malicious.

15.2.2 One-Class Feature Selection

Feature selection techniques choose a subset of original features to reduce the dimension of the input. One of the goals to perform feature selection is to remove redundant, irrelevant or trivial information in the original set of features. By removing some irrelevant features, the classification performance may be improved. It is also usually used as a very important pre-processing step for high dimensional data to reduce the computational complexity.

There are two types of feature selection methods: wrapper and filter. Wrapper methods are useful for feature selection of supervised learning algorithms which have labels of all classes while filter methods can be applied to any type of dataset regardless of learning methods nor data labels. In addition, filter methods are generally faster than wrapper ones. Thus, filter methods are appropriate for one-class feature selection. In the following, we introduce three filter based one-class feature selection algorithms.

15.2.2.1 Intra-Class Distance (ICD)

ICD is defined as the mean L_p distance of all samples to the centroid of the class, as shown in Eq. (15.1) [7] where n is the number of samples, $\boldsymbol{f_s}$ is the selected feature set, $\boldsymbol{x_i}$ is the feature vector of ith sample, and $\boldsymbol{\mu}$ is the centroid of the class. Lower intra-class distance indicates that samples in the class are more similar to each other. When used for one-class feature selection, the reduction of intra-class distance due to the removal of a feature is used as the measure of its importance. Larger reduction from removing a feature will give this feature a higher ranking to be selected.

$$ICD(\boldsymbol{f_s}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x_i} - \boldsymbol{\mu}\|_p \tag{15.1}$$

In this paper, we use the Manhattan distance (L_1 norm) to calculate *ICD* instead of the Euclidean Distance (L_2 norm) used in [7] to achieve a lower computational complexity in feature selection.

15.2.2.2 Pearson Correlation Coefficient (PCC)

PCC is used in statistics to measure the linear correlation between two variables [8]. The formula to calculate the

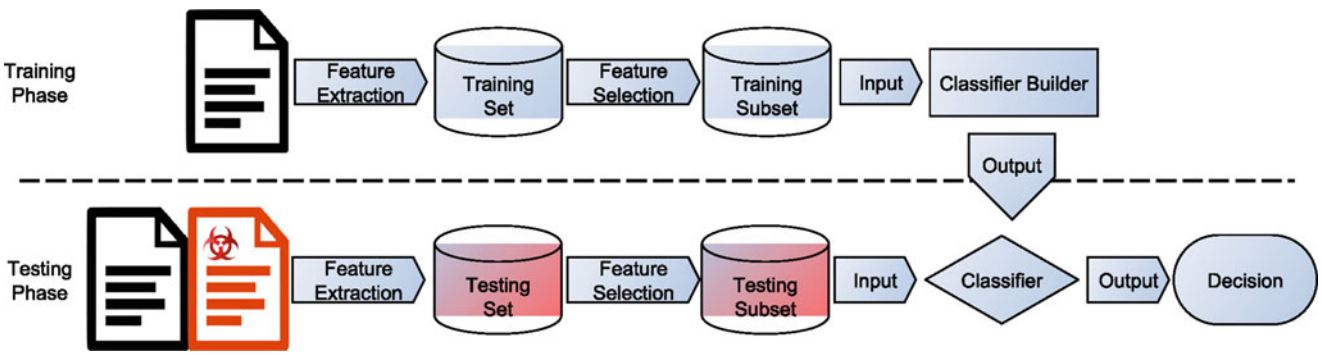

Fig. 15.1 Zero-day Andorid malware detection using one-class feature selection and classification

correlation of two features $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$ is shown in Eq. (15.2), where $cov(\boldsymbol{f_i}, \boldsymbol{f_j})$ is the covariance of features $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$, $\boldsymbol{\sigma_i}$ and $\boldsymbol{\sigma_j}$ are the standard deviations of $\boldsymbol{f_i}$ and $\boldsymbol{f_j}$, respectively. The range of PCC value is between -1 and $+1$, which indicates the linear correlation from total negative to total positive. A zero value indicates no linear correlation.

$$\rho_{i,j} = \frac{\text{cov}\left(\boldsymbol{f_i}, \boldsymbol{f_j}\right)}{\sigma_i \sigma_j} \tag{15.2}$$

When used for one-class feature selection [7], the PCC value of a feature $\boldsymbol{f_i}$ is calculated as the sum of the absolute values of its pearson correlations to other features as shown in Eq. (15.3). A lower PCC value of a feature will give it a higher chance to be selected.

$$PCC\left(\boldsymbol{f_i}\right) = \sum_{j=1, j\neq i}^{m} \left|\rho_{i,j}\right| \tag{15.3}$$

15.2.2.3 Laplacian Score (LS)

LS is based on the observation that data points from the same class are close to each other [9]. Therefore, features can be evaluated according to their power of locality preserving. *LS* based one-class feature selection algorithm first constructs a nearest neighbor graph with each sample as a node. An edge, $\boldsymbol{S}_{i,j}$ defined in Eq. (15.4), where t is a constant, will be added between two nodes, $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, if they are close enough, i.e. at least one of them is among k nearest neighbors of the other one. Otherwise, $\boldsymbol{S}_{i,j} = 0$. The constructed weight matrix $\boldsymbol{S}$ models the local structure of the data space.

$$\boldsymbol{S}_{i,j} = e^{-\frac{\left\|\boldsymbol{x_i}-\boldsymbol{x_j}\right\|^2}{t}} \tag{15.4}$$

The *LS* score for the ith feature $\boldsymbol{f_i}$ is calculated using Eq. (15.5) where $\boldsymbol{D}$ and $\boldsymbol{L}$ are the corresponding degree matrix and Laplacian matrix of $\boldsymbol{S}$. Features with larger *LS* values are more significant.

$$\begin{aligned} LS\left(\boldsymbol{f_i}\right) &= \frac{\tilde{\boldsymbol{f}}_i^T \boldsymbol{L} \tilde{\boldsymbol{f}}_i}{\tilde{\boldsymbol{f}}_i^T \boldsymbol{D} \tilde{\boldsymbol{f}}_i} \\ \tilde{\boldsymbol{f}}_i &= \boldsymbol{f}_i - \frac{\boldsymbol{f}_i^T \boldsymbol{D} \boldsymbol{1}}{\boldsymbol{1}^T \boldsymbol{D} \boldsymbol{1}} \boldsymbol{1} \end{aligned} \tag{15.5}$$

15.2.3 One-Class Classification

Unlike two-class classification, OCC only uses samples from the target class for training, which poses challenges for building the classification model. For Android malware detection, we define the benign apps as target class samples and the malware as outliers. The OCC classifier is modeled as a function $f(\boldsymbol{x})$ that accepts an input sample $\boldsymbol{x}$ and yields a quantitative value which could be a distance, probability or other metrics. Then a decision function $h(\boldsymbol{x})$ defined in Eq. (15.6) is used to determine the input sample as target class ($h(\boldsymbol{x}) = 0$) or outlier ($h(\boldsymbol{x}) = 1$) based on a threshold θ. The threshold is affected by the choice of a parameter called outlier ratio, R. Outlier ratio is used to reduce the effect of outfitting. A larger outlier ratio value may accept more target class samples as well as outliers, while a smaller one may reject more samples from both classes.

$$h\left(\boldsymbol{x}\right) = \begin{cases} 0 & \text{if } f\left(\boldsymbol{x}\right) \leq \theta \\ 1 & \text{if } f\left(\boldsymbol{x}\right) > \theta \end{cases} \tag{15.6}$$

In this paper, we tested the following OCC methods for Android malware detection.

15.2.3.1 Gauss Distribution

Gauss Distribution is a classifier that models the target class samples as Gaussian distribution [10]. The m-dimensional Gauss probability distribution is shown in Eq. (15.7).

$$p_N\left(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{(2\pi)^{m/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{15.7}$$

where $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix, and m is the number of features.

The Mahalanobis distance calculated using Eq. (15.8), is then used to measure the distance from a sample $\boldsymbol{x}$ to the modeled distribution. Based on the outlier ratio (R), a threshold θ is determined as the $(n \times R)$th largest Mahalanobis distance in the training set. The new sample will then be classified as target class or outlier based on Eq. (15.6).

$$f\left(\boldsymbol{x}\right) = \left(\boldsymbol{x} - \boldsymbol{\mu}\right)^T \boldsymbol{\Sigma}^{-1} \left(\boldsymbol{x} - \boldsymbol{\mu}\right) \tag{15.8}$$

15.2.3.2 K-Means

K-Means is an unsupervised clustering algorithm [10]. The data is grouped into k clusters while the average distance to a cluster centroid $\boldsymbol{c_i}$ is minimized. The centroid of each cluster is found in an iterative way which is arbitrary initialized in the beginning of the algorithm. In each iteration, a training sample is assigned to a cluster whose centroid is the closest to the sample. Once all samples are assigned to clusters, the centroid of each cluster will be recomputed as the mean of all the samples in the cluster. The procedure will be repeated until the clusters are stable. When K-Means is used for one-class classification, the target class is characterized by Eq. (15.9) and the output is calculated with Eq. (15.6).

$$f\left(\boldsymbol{x}\right) = \min_i (\boldsymbol{x} - \boldsymbol{c}_i)^2 \tag{15.9}$$

15.2.3.3 Principle Component Analysis (PCA)

PCA [11] projects the target data to a new linear subspace, which is defined by k eigenvectors of the data covariance

matrix, i.e. given a $m \times m$ data covariance matrix $\boldsymbol{\Sigma}$, we only use k eigenvectors with largest eigenvalues and define this sub-matrix as $\boldsymbol{W}$. Reconstruction error is computed to determine if a testing sample fits into the target subspace.

$$f(\boldsymbol{x}) = \left\| \boldsymbol{x} - \boldsymbol{x}_{proj} \right\|^2 \tag{15.10}$$

where the projection is computed as

$$\boldsymbol{x}_{proj} = \boldsymbol{W}\left(\boldsymbol{W}^T \boldsymbol{W}\right)^{-1} \boldsymbol{W}\boldsymbol{x} \tag{15.11}$$

15.2.3.4 *k*-Nearest Neighbor Data Description (KNNDD)

KNNDD is based on Nearest Neighbor Data Description (NNDD) method, in which a sample is tested by comparing its local density with its nearest neighbor local density. KNNDD replaces the nearest neighbor with the kth nearest neighbors. The acceptance function of KNNDD is shown in Eq. (15.12). A sample $\boldsymbol{x}$ will be classified as the target class if the fraction of the distance between $\boldsymbol{x}$ and its kth nearest neighbor $NN_k^{tr}(\boldsymbol{x})$to the distance between the kth nearest neighbor and its kth nearest neighbor is less than or equal to a threshold θ.

$$f(\boldsymbol{x}) = \frac{\left\| \boldsymbol{x} - NN_k^{tr}(\boldsymbol{x}) \right\|}{\left\| NN_k^{tr}(\boldsymbol{x}) - NN_k^{tr}\left(NN_k^{tr}(\boldsymbol{x})\right) \right\|} \tag{15.12}$$

15.2.3.5 *ν*-SVM

Support vector machine (SVM) is a popular classification method for machine learning. However, original SVM can only be used for supervised learning problem, i.e. multiple-class problem. ν-SVM was proposed in [12] for one-class problem, which looks for a boundary that accepts the target class samples and rejects outliers. Ideally, outliers will be located outside that boundary. However, the boundary may have overfitting problem. In practice, the boundary could be shrinked so that some target class samples will be located outside the boundary. ν-SVM maps the data into another feature space of higher dimension, then calculates a hyperplane that separates the mapped target samples from the origin with maximal margin. The hyperplane is then used as the boundary to accept target class samples while reject outliers. The quadratic problem to be solved is defined as below, where $\boldsymbol{\Phi}$ is the feature map function which maps a sample to another hyperspace, the fraction of training errors and regularization $||\boldsymbol{w}||$ are controlled by ν; $\boldsymbol{\xi}$ is for penalization.

$$\begin{aligned} &\min_{\boldsymbol{w}\in\boldsymbol{F},\boldsymbol{\xi}\in\boldsymbol{x}^l,\rho\in\boldsymbol{x}} \tfrac{1}{2}\|\boldsymbol{w}\|^2 + \tfrac{1}{\nu n}\sum_i \xi_i - \rho \\ &\text{subject to} \\ &(\mathbf{w}\cdot\boldsymbol{\Phi}(\boldsymbol{x}_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{15.13}$$

After solving the problem with $\boldsymbol{w}$ and ρ, the decision function is define as

$$f(\boldsymbol{x}) = \operatorname{sgn}\left((\boldsymbol{w}\cdot\Phi(\boldsymbol{x})) - \rho\right) \tag{15.14}$$

15.2.3.6 Minimax Probability Machine (MPM)

One-class MPM was proposed in [13] for novelty detection. This method uses mean and covariance matrix of the distribution to minimize the worst case probability of data points falling outside of the convex set. The problem of one-class MPM is defined in Eq. (15.15) to find a half-space Q that $\Pr\{\boldsymbol{x} \in Q\} = \alpha$, where $a \in \mathbf{R}^n \backslash \{0\}$, $b \in \mathbf{R}$, probability at least α, $\alpha \in (0, 1)$, for every distribution having mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are bounded in a set $\boldsymbol{X}$.

$$\inf_{x\sim(\boldsymbol{\mu},\boldsymbol{\Sigma})} \Pr\left\{\mathbf{a}^T\boldsymbol{x} \geq b\right\} \geq \alpha, \forall(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \boldsymbol{X} \tag{15.15}$$

Once the optimal decision region is determined, the decision function is defined as:

$$f(\boldsymbol{x}) = \mathbf{a}^T\phi(\mathbf{x}) = \sum_{i=1}^{n} \boldsymbol{\gamma}_i K(\boldsymbol{x}_i, \boldsymbol{x}) \tag{15.16}$$

where φ is a mapping function, $K(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is the kernel function, $\boldsymbol{\gamma}$ is a parameter determined during training. Equation (15.6) is used for classification with threshold given by b.

15.3 Performance Evaluation

To evaluate the performance of various one-class feature selection and classification methods tested for malware detection, we used the Drebin dataset [14] which includes 5560 malware samples and 123,453 benign samples. We chose 5467 features of six main categories from the dataset including Restricted API Call, Suspicious API Call, Hardware Component, Requested Permission, Used Permission and Intent.

For the purpose of performance evaluation, the Drebin dataset was divided into tenfolds of training set and testing set. Each fold of testing set includes all 5560 malware samples of the Drebin dataset. To prepare a balanced testing set, each fold of testing set also includes randomly picked 5560 benign apps without overlapping with other folds. The rest of the benign apps were randomly divided and assigned into tenfolds as the training sets.

The performance matrics used for evaluation are sensitivity, specificity, and balanced accuracy which are calculated as follows:

$$Sensitivity = \frac{TP}{TP + FN} \tag{15.17}$$

$$Specificity = \frac{TN}{TN + FP} \tag{15.18}$$

$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2} \tag{15.19}$$

where *TP*, *TN*, *FP*, *FN* are true positives, true negatives, false positives and false negatives, respectively.

15.3.1 Performance Comparison of Feature Selection Methods

We first tested the classification performance of the three feature selection methods. Figures 15.2 and 15.3 show the results of using *Gauss Distribution* and ν-SVM as the classifiers, respectively. The outlier ratio, *R*, was set as 0.2. The results showed that *ICD* and *LS* have comparable classification performance while *PCC* has a significant worse performance than the other two methods. The performance of *ICD* and *LS* is stabilized when the number of features reaches 150, where the performance is comparable to the performance of using all features.

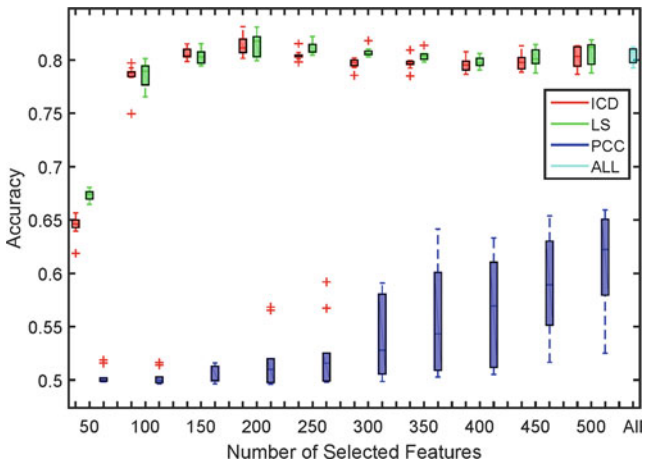

Fig. 15.2 Classification results of using different one-class feature selection methods (*Gauss Distribution*)

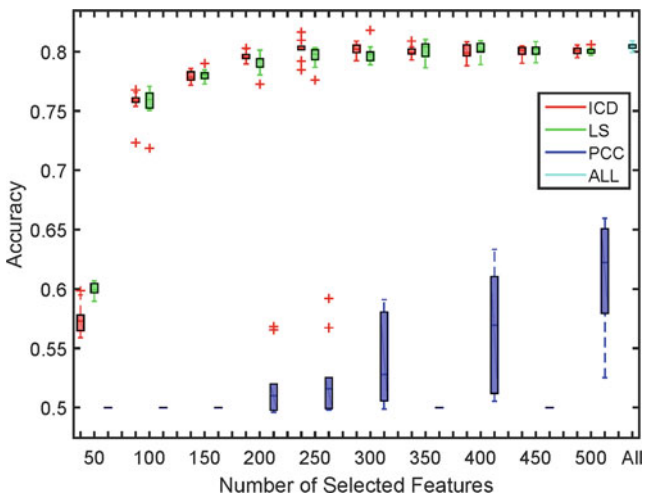

Fig. 15.3 Classification results of using different one-class feature selection methods (ν-SVM)

We also compared the mean and standard deviation of computation time used by each feature selection method and the results are shown in Table 15.1. The experiment was performed on a server with an Intel Xeon CPU X5660, 2.8 GHz, four cores and 16 GB memory. Obviously, ICD uses significantly lower computation time compared with other two methods.

Since *ICD* and *LS* achieve comparable performance, we also investigated the difference between the features selected by these two methods. Figure 15.4 shows the percentage of features selected by *ICD* and *LS* that are different. For the cases that the number of features is <200 or more than 400, the percentage of difference is around or <5%. The largest difference is <20% when the number of selected features is 300. The results showed that the sets of features selected by *ICD* and *LS* are similar to each other. Considering the similarity of selected features, the comparable classification performance and the computational complexity, *ICD* is the best one among the three tested methods for selecting features for Android malware detection.

Table 15.1 Computation time used by each feature selection method in seconds

Method	Computation time
ICD	0.094 (0.007)
PCC	1165.904 (25.719)
LS	13.572 (0.371)

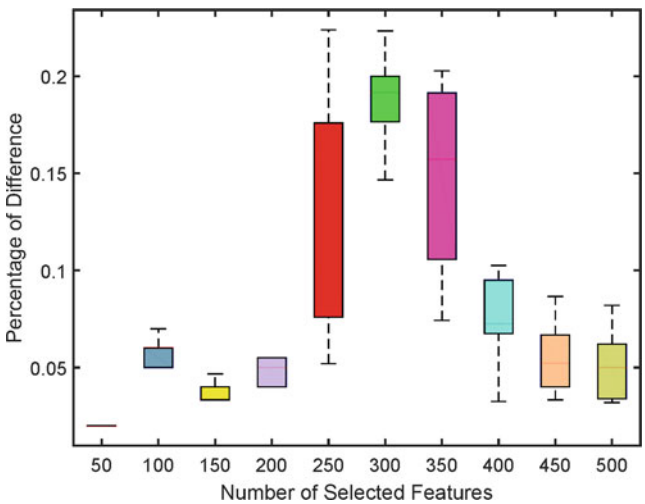

Fig. 15.4 Difference between the features selected by ICD and LS

15.3.2 Results of Classification

By using ICD as the feature selection method, we tested the performance of the six selected one-class classifiers under different outlier ratios. The classifiers were implemented with the Matlab toolbox *Dd_tools* [15] and Python *Scikit-learn* [16]. The number of selected feature was fixed as 200. The classification results are shown in Table 15.2. In each cell of the table, we show the mean and standard deviation of sensitivity, specificity and accuracy from top to bottom. It can be seen that *Gauss* consistently has the best performance in terms of the classification accuracy. Although MPM has very high sensitivity but the low specificity makes it not a good choice.

15.3.3 One-Class Classification vs. Two-Class Classification

Two-class classification methods are typically more popular than OCC methods since they can achieve better performance when a large amount of representative samples are available for training. However, lack of representative samples may lead to serious overfitting problem when using two-class classification methods. Therefore, we compared the performance of one-class classification with that of two-class classification for Android malware detection. Two OCC methods, *Gauss* and ν-SVM, and two popular two-class classification methods, SVM and Classification And Regression Tree (*CART*), were used for comparison. The outlier ratio for the OCC methods was set to 0.2. In the experiment, we only used the largest nine malware families from Drebin dataset and each of them has more than 100 samples. These nine families were sorted in descending order by each family's distance to all other families according to Eq. (15.20), where d_i is the distance for the ith family, n_i, n_j are the numbers of samples in the ith and jth families respectively, $\boldsymbol{x}_{ri}$ and $\boldsymbol{x}_{rj}$ are the r_ith sample from the ith family and the r_jth sample from the jth family respectively. A larger d_i means that the behavior of the ith family is more different from those of other families. The distances and sizes of the families are listed in Table 15.3.

Table 15.3 Ranking of malware families by distance to other families

Family name	Distance	No. of samples
FakeDoc	432.74	132
Plankton	350.74	625
DroidKungFu	330.82	667
Iconosys	310.59	152
GinMaster	306.17	339
BaseBridge	303.74	330
Kmin	300.79	147
FakeInstaller	298.75	925
Opfake	296.07	613

$$d_i = \sum_{j=1, j\neq i}^{9} \left(\frac{1}{n_i \times n_j} \sum_{r_i=1}^{n_i} \sum_{r_j=1}^{n_j} \left| \mathbf{x}_{r_i} - \mathbf{x}_{r_j} \right| \right) \tag{15.20}$$

We divided the benign apps equally into tenfolds. In each fold, half of the benign apps were used for training of OCC methods. The two-class classifiers were trained using this half of benign apps with the bottom-five malware families in

Table 15.2 Classification results

	Performance metric	0.10	0.15	0.20	0.25
Gauss	Sensitivity	0.694 (±0.015)	0.787 (±0.015)	0.833 (±0.017)	0.878 (±0.003)
	Specificity	0.891 (±0.006)	0.843 (±0.005)	0.794 (±0.005)	0.746 (±0.004)
	Balanced accuracy	**0.793 (±0.006)**	**0.815 (±0.007)**	**0.814 (±0.009)**	**0.812 (±0.003)**
K-mean	Sensitivity	0.538 (±0.014)	0.663 (±0.011)	0.738 (±0.013)	0.802 (±0.015)
	Specificity	0.903 (±0.003)	0.853 (±0.004)	0.803 (±0.006)	0.751 (±0.006)
	Balanced accuracy	0.721 (±0.007)	0.758 (±0.005)	0.771 (±0.007)	0.776 (±0.007)
KNN	Sensitivity	0.634 (±0.020)	0.730 (±0.022)	0.801 (±0.023)	0.844 (±0.011)
	Specificity	0.903 (±0.003)	0.863 (±0.006)	0.812 (±0.004)	0.773 (±0.010)
	Balanced accuracy	0.769 (±0.010)	0.796 (±0.011)	0.806 (±0.012)	0.808 (±0.007)
PCA	Sensitivity	0.666 (±0.019)	0.746 (±0.016)	0.821 (±0.018)	0.870 (±0.010)
	Specificity	0.895 (±0.003)	0.844 (±0.004)	0.793 (±0.004)	0.743 (±0.005)
	Balanced accuracy	0.781 (±0.009)	0.795 (±0.007)	0.807 (±0.009)	0.806 (±0.007)
ν-SVM	Sensitivity	0.575 (±0.009)	0.711 (±0.008)	0.790 (±0.005)	0.853 (±0.004)
	Specificity	0.903 (±0.003)	0.852 (±0.004)	0.803 (±0.005)	0.753 (±0.007)
	Balanced accuracy	0.739 (±0.005)	0.782 (±0.004)	0.796 (±0.004)	0.803 (±0.005)
MPM	Sensitivity	0.980 (±0.004)	0.981 (±0.005)	0.981 (±0.004)	0.983 (±0.004)
	Specificity	0.423 (±0.006)	0.418 (±0.007)	0.414 (±0.007)	0.408 (±0.006)
	Balanced accuracy	0.701 (±0.004)	0.699 (±0.004)	0.698 (±0.004)	0.696 (±0.004)

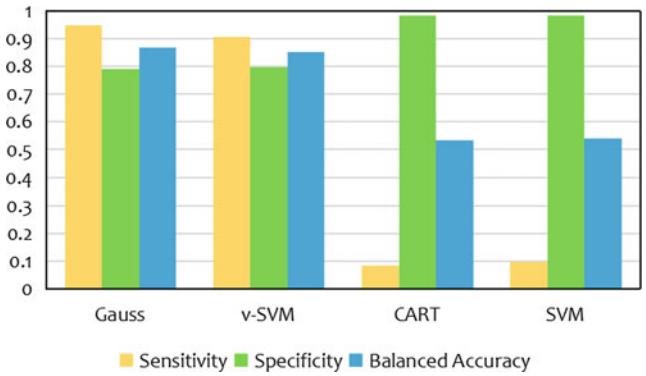

Fig. 15.5 Performance comparison of one-class classification and two-class classification

Table 15.3. The other half of the benign apps and the top-four malware families were combined as the testing set. Feature selection was performed for all methods and the number of selected features was 200. ICD was used for the OCC methods and χ^2 test was used for the two-class classification methods.

Figure 15.5 shows the detection performance in terms of sensitivity, specificity and balanced accuracy, respectively. The reported number is the averaging of tenfolds. It can be seen that OCC methods achieve significantly better sensitivity and balanced accuracy than two-class classification methods. We checked two-class classifiers and found that they have excellent performance on training set: *SVM* and *CART* achieve 97.4% and 99.9% in terms of sensitivity, 97.9% and 99.9% in terms of average balance accuracy, respectively. The results indicate that the detection models trained using two-class classification methods suffer serious overfitting problem when lack of representative samples of malware families for training while OCC methods are capable of detecting zero-day malware attacks.

15.4 Conclusion

In this paper, we investigated the use of one-class feature selection and classification for zero-day Android malware detection. By using these approaches, we do not need to utilize malicious samples for training which makes it suitable for the task. Different one-class feature selection and classification methods were tested and compared. The results showed that *ICD* is the best method for feature selection in terms of classification performance and computational complexity and *Gauss* is the best method for classification in terms of the balanced accuracy. We also demonstrated the capability of OCC methods over two-class classification methods in detecting zero-day malware attacks.

Acknowledgements The work of Jun Zheng was supported in part by the National Science Foundation under EPSCoR Cooperative Agreement OIA-1757207.

References

1. Liu, X., Liu, J.: A two-layered permission-based android malware detection. In: 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 142–148. IEEE (2014)
2. Aager, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in android. In: International Conference on Security and Privacy in Communication Systems, pp. 86–103. Springer, Basel (2013)
3. Wang, Y., Watson, B., Zheng, J., Mukkamala, S.: ARP-miner: mining risk patterns of android malware. In: International Workshop on Multi-disciplinary Trends in Artificial Intelligence, pp. 363–375. Springer, Basel (2015)
4. Sahs, J., Khan, L.: A machine learning approach for andorid malware detection. In: 2012 European Intelligence and Security Informatics Conference, pp. 141–147. IEEE (2012)
5. Guo, X., Yin, Y., Dong, C., Yang, G., Zhou, G.: On the class imbalance problem. In: ICNC '08: Proceedings of the 2008 Fourth International Conference on Natural Computation, pp. 192–201. IEEE (2008)
6. Tax, D.: One class classification. PhD thesis, Delft University of Technology (2001)
7. Lorena, L., Carvalho, A., Lorena, A.: Filter feature selection for one-class classification. J. Intell. Robot. Syst. **80**, 227–243 (2015)
8. Benesty, J., Chen, J., Huang, Y., Cohen, I.: Pearson correlation coefficient. In: Noise Reduction in Speech Processing, pp. 1–4. Springer, Berlin (2009)
9. He, X., Cai, D., Niyogi, P.: Laplacian score for feature selection. In: NIPS'05: Proceedings of the 18th International Conference on Neural Information Processing Systems, pp. 507–514. MIT Press, Cambridge (2005)
10. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press, New York (1995)
11. Tax, D., Müller, K.: Feature extraction for one-class classification. In: Artificial Neural Networks and Neural Information Processing — ICANN/ICONIP 2003. ICANN 2003, ICONIP 2003, vol. 2003, pp. 342–349. Springer, Berlin (2003)
12. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. In: NIPS'99: Proceedings of the 12th International Conference on Neural Information Processing Systems, pp. 582–588. MIT Press, Cambridge (1999)
13. Ghaoui, L., Jordan, M., Lanckriet, G.: Robust novelty detection with single-class MPM. In: Advances in Neural Information Processing Systems, pp. 929–936. NIPS Foundation (2003)
14. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: effective and explainable detection of android malware in your pocket. In: NDSS'14: Network and Distributed System Security Symposium. NDSS (2014)
15. Tax, D.: Dd_tools - the data description toolbox for Matlab (2015)
16. Pedregosa, F., Varoquaux, G., Gramfort, A., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)