Android Malware Detection Using Ensemble Learning on Sensitive APIs

Junhui Yu[1,2(✉)], Chunlei Zhao[1,2], Wenbai Zheng[1,2], Yunlong Li[1,2], Chunxiang Zhang[1,2], and Chao Chen[1,2]

[1] Key Laboratory of Computer Vision and System, Ministry of Education, Tianjin University of Technology, Tianjin 300384, China
[2] Tianjin Key Laboratory of Intelligence Computing and Novel Software Technology, Ministry of Education, Tianjin University of Technology, Tianjin 300384, China

Abstract. In recent years, with the quiet popularity of mobile payment methods, mobile terminal equipment also have potential security problems while facilitating people's lives. Behavior-based Android malware detection is mostly based on permission analysis and API calls. In this paper, we propose a static Android malicious detection scheme based on sensitive API calls. We extracted all APIs called in the experimental samples through decompilation, and then calculated and ranked the threats related to these APIs according to the mutual information model, selected the top 20 sensitive API calls, and generated a 20-dimensional feature vector for each application. In the classification process, an integrated learning model based on DT classifier, kNN classifier and SVM classifier is used to effectively detect unknown APK samples. We collected 516 benign samples and 528 malicious samples. Through a large number of experiments, the results show that the accuracy of our scheme can be up to 94%, and the precision is up to 95%.

Keywords: Android · Sensitive API · Mutual information · Malware detection

1 Introduction

The market share of smartphones running the Android operating system will rise from 85.1% in 2018 to 87% [1]. The strong compatibility of the Android system also attracts more and more developers. At the same time, due to the lack of strong detection mechanisms and processes in the Google [2], this provides a wide range of possibilities for the release and promotion of malicious applications.

The current behavior-based Android malware detection is mainly divided into two parts. One is dynamic malware detection technology that simulates running in sandboxes or virtual machines, and the other is static malware detection technology that extracts relevant features through reverse engineering. Static analysis matches specific characteristics of known malicious applications to detect

Supported by Tianjin Nature Science Youth Foundation (No.18JCQNJC69900).

H. Jiang et al. (Eds.): ICECI 2020, LNICST 368, pp. 126–140, 2021.
https://doi.org/10.1007/978-3-030-73429-9_8

whether the sample is malware. However, with the development and changes of malicious applications, the efficiency of a single static detection is not very high, and there are certain false positives and false negatives. At the same time, the dynamic analysis of Android malware can provide real-time and comprehensive detection when the application is running, but there are still higher requirements in terms of statistical information and the deployment of the detection environment. Moreover, the dynamic detection technology cannot simultaneously detect a large number of applications simultaneously.

However, there are obvious differences between malicious applications and normal applications in corresponding API function calls. This paper uses sensitive API functions as features to detect malicious application software.

This paper proposes a static Android malicious detection method based on integrated learning. The method is mainly divided into three parts: feature extraction stage, training stage and detection stage. For the feature extraction stage, the mutual information model is used to generate 20-dimensional API feature vectors ranked by sensitivity level; during the training stage, the feature vectors are input to the associated three basic classifiers (decision tree(DT)classifier, k nearest neighbor (kNN) classifier and support vector machine (SVM) classifier) to generate the final training results. In the detection phase, the main task is to quickly classify unknown APKs by using an integrated learning model. A large number of experimental results show that the accuracy of the scheme can reach 94%, and the precision is up to 95%.

The specific experimental content of this paper is as follows:

- We collect top20 sensitive API calls as detection features to improve detection accuracy.
- By comparing the detection performance of different combinations of various classification algorithms, we propose an integrated learning algorithm based on DT classifier, kNN classifier and SVM classifier as the basic classifier.
- Experiments show that this method can reduce the complexity of Android malware detection, and improve detection accuracy.

The rest of the paper is organized as follows: Sect. 2 introduces the related work, including the analysis of the research status of dynamic malware detection technology and static malware detection technology. Sections 3 and 4 introduce the experimental design of the feature extraction and detection process in this paper. Sections 5 and 6 give the results and conclusions of the experiment.

2 Related Work

2.1 Dynamic Detection Method

With the evolution and development of malicious applications, more and more malicious applications evade the corresponding static detection by obfuscating code or hardening protection. This led to a situation where the false detection rate of Android static detection was slightly higher under certain conditions.

Dynamic program behavior monitoring technology can record the behavior of the program during dynamic execution, so as to avoid the confusion of static code by malicious programs. Depending on the recording granularity, information such as instruction sequence, system call, API sequence, and API parameters can be recorded. According to the principle of program behavior monitoring technology, it can be divided into three categories: program behavior monitoring technology based on binary instrumentation, program behavior monitoring technology based on virtual machine and program behavior monitoring technology based on Hook.

Dynamic program behavior monitoring technology can record the behavior of the program during dynamic execution, so as to avoid the confusion of static code by malicious programs. TaintDroid [3] is an efficient system-level dynamic stain tracking and analysis system that can track multiple sensitive data sources at the same time, providing users with a view of the use and sharing of private data by third-party applications. TaintDroid is deployed on mobile phones, so its overhead cannot be ignored. Crowdroid [4] is a dynamic analysis method. It obtains the application behavior logs from the user's mobile phone through crowdsourcing. The specific data collected is the system call, and the data is uploaded to the remote server, and the data is clustered through K-means. In order to find malicious applications, it is difficult to apply them without incentives.

2.2 Static Detection Method

The most obvious feature of the static detection technology based on reverse engineering is that it does not need to execute the malicious application to obtain the relevant static features contained in the application by decompilation, including the permission information applied by the application, the API function calls, etc. Wu et al. [5] proposed an Android malware detection system, which uses a dataflow application program interfaces (APIs) as classification features to detect Android malware. This paper uses a thorough analysis, to extract dataflow-related API-level features, and to improve the kNN classification model, and uses machine learning to further optimize the API list dataflow-related and improve the detection efficiency. The accuracy of the system in detecting unknown Android malware is as high as 97.66%. Onwuzurike, L et al. [6] proposed a MAMADROID model based on a static analysis system is proposed, which uses the sequence obtained by the application's API call graph (such as Markov chain) to build the model to ensure that the model's changes to the API and the size of the feature set are convenient for management. The MAMADROID model has a high detection rate, but still has a certain FPR. Kim, T et al.[7] proposes A novel Android malware detection framework. The framework extracts multiple features such as permissions and API calls, and refines these features to achieve an effective feature representation for malware detection. In addition, a multi-modal deep learning method is proposed as a malware detection model. It was found that the detection method greatly improves

the detection rate, but the complexity is high. Zhao et al.[8]propose a detection method that uses sensitive APIs as detection features. But they used two basic classifiers, the detection accuracy only to 92%.

3 Feature Extraction and Analysis

Based on the in-depth understanding of Android malware detection in this paper, you can extract the information of sensitive API calls and analyze based on this feature, which mainly includes the following 3 steps:

- Feature generation: decompilation of the application program of the experimental sample to extract all API call information.
- Generate a collection of sensitive API calls: use the mutual information model to calculate and rank the correlation between each API call and malware in the sample.
- Generate feature vectors: Select the top 20 sensitive API calls to generate a 20-dimensional feature vector for each application.

3.1 Preliminaries

In the early stage of this paper, a lot of information collection and data mining were carried out on the Android operating system, mainly including API features, sensitive API calls, and the selection of mutual information models.

API Features. The Android application is released in the form of an APK, which is essentially a compressed package file. The corresponding file obtained by decompressing APK is shown in Fig. 1.

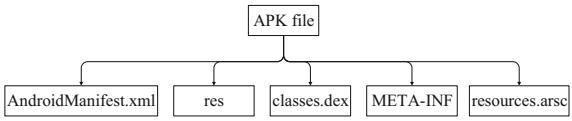

Fig. 1. APK file structure

Among them, the android manifest.xml file is used to store the package name of Android application, permissions, SDK version number, etc. It is the configuration file of Android application package (APK). The res file is a storage resource file in which the images, strings, layouts, etc. of Android APK are stored. The META-INF file is used to hold the signature information of APK to ensure the integrity of APK. The classes.dex is a Java bytecode file that can be directly run in a Dalvik virtual machine.

Due to the good adaptability of the APIs, the operating system can provide service interfaces for various types of applications. When the request of data

access, network data connection, file read and write, or other important resources appear, the applications will invoke the APIs. Because of the important role of the APIs, malicious applications will call the target APIs to achieve some malicious behavior.

Sensitive API Call. In the Android platform, there is a correspondence between permissions and API calls to access specific resources of the system. The Android platform API list describes nearly 8000 calling methods. Among them, there is a specific correspondence between certain APIs and permissions. For example, when an application calls the SendMessage() API for sending short messages, the Android system process checks whether the application has applied for permission to send short messages: Android.permission.SEND_SMS. Only when the application has applied for the permission and the permission is declared in the manifest file $<Manifest>$ will the SendMessage() API be called when it is used. Android's official website lists all the permission information, mainly divided into four categories: normal, dangerous, signature, signatureorSystem. The permission risks they represent increase in turn, which means that the risk of the API corresponding to them also increases in turn.

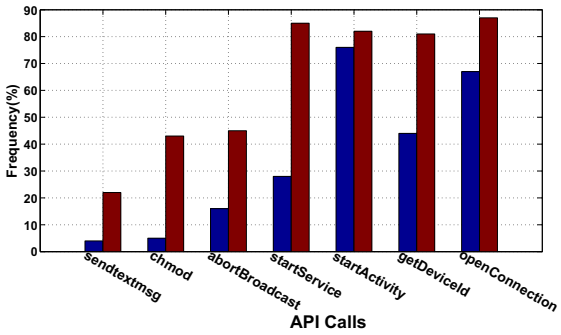

Fig. 2. Frequency of calling certain APIs in malware and benign applications

As shown in Fig. 2, we found that the frequency of calling certain APIs is quite different between malicious applications and normal applications through experiments. Therefore, these sensitive APIs can be used as one of the basis for identifying the maliciousness of the application. In this study, we focus on high-risk APIs that involve sensitive user data. Hanna [9] have studied the permission mechanism in depth and given the corresponding relationship between the permissions and APIs. According to the APIs corresponding to the sensitive permission list, the mutual information model is used to measure the relationship between API and application maliciousness.

Mutual Information Model. In machine learning, the correlation between measured features and class variables is called feature ranking, and its purpose is to select the features with the largest amount of information and improve the performance of the learning model [11]. In probability theory and information theory, Mutual Information (MI) [10] for two random variables is a measure of the interdependence between variables. Unlike correlation coefficients, mutual information is not limited to real-valued random variables. It is suitable for more general and common application scenarios and determines the product p(x) of the joint distribution p(X,Y) and the decomposed edge distribution. The similarity of p(y). Mutual information can be used to measure the mutual dependence between two sets of events. The mutual information calculation formula of two discrete random variables X and Y is shown as formula (1):

$$I(X,Y) = \sum_{x_i} \sum_{y_j} p(X = x_i, Y = y_j) \times log \frac{p(X = x_i, Y = y_j)}{p(X = x_i) \times p(Y = y_j)} \quad (1)$$

P(x, y) is the joint probability distribution function of X and Y, while p(x) and p(y) are marginal probability distribution functions of X and Y respectively. In formula (1), the variable X represents that the APIs exist in an Android application software (or not), variable Y on behalf of the application of categories (application belongs to malicious applications or benign applications). P(x) means the probability that the variable X is equal to x, p(y) means the probability that the variable Y is equal to y. P(x, y) is the value of x with respect to X, and the value of Y is the probability of y. In the case of continuous random variables, the summation formula is replaced by the double integral formula, which is shown in the following formula (2):

$$I(x,y) = \int_Y \int_X p(x,y) \times log(\frac{p(x,y)}{p(x) \times p(y)}) dxdy \quad (2)$$

Similar to formula (1), in Eq. (2), p(X, Y) still represents the joint probability distribution function of X and Y, while p(x) and p(y) are the marginal probability distribution functions of X and Y, respectively. Mutual information is the inherent dependence between the joint distribution of X and Y relative to the joint distribution under the assumption that X and Y are independent. So mutual information measures dependency in the following way: $I(X,Y) = 0$ if and only if X and Y are independent random variables. It is easy to see from one direction: when X and Y are independent, $p(x,y) = p(x) \cdot p(y)$, so we can draw the conclusion shown as formula (3):

$$log(\frac{p(x,y)}{p(x) \times p(y)}) = log1 = 0 \quad (3)$$

In addition, mutual information is non-negative ($I(X;Y) \geq 0$) and symmetrical ($I(X,Y) = I(Y,X)$).

3.2 Feature Generation

Feature Extraction. In this paper, we use the script file of the Androguard tool to statically analyze the experimental samples. Figure 3 shows the Android API call examples of some experimental samples obtained.

```
java/lang/Long intValue()I
java/lang/Long <init>(J)V
java/lang/LongvalueOf(J)Ljava/lang/Long;
android/util/StateSet stateSetMatches ([I[I)Z
android/util/StateSet trimStateSet([I[I)[I
android/view/ViewTreeObserver$InternalInsetsInfo <init>()V
```

Fig. 3. Android API call sequence

Generate a Collection of Sensitive API Calls. Mutual information can be used to measure the correlation between two sets of events. This paper uses an effective sorting method, that is, "mutual information" as shown in formula (1), to measure the correlation between Android applications and specific API calls, and based on the calculated correlation, a set of sensitive APIs is generated. And the sensitive value of each API is calculated separately.

The final calculation result is between 0 and 1. The larger the value of the result, the higher the correlation between the two. If the value is 1, it means that the two are necessarily related, and similarly, the value is 0 that the two are not related. Extract 20 API functions most relevant to malicious applications as a collection of sensitive API calls.

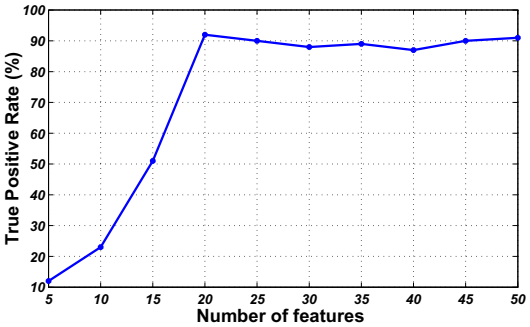

Fig. 4. The TPR of different number of features on detection results

As shown in Fig. 4, the reason why we choose the top 20 APIs for sensitivity calculation as the reference standard for selecting functional APIs is because if

the number of selected APIs is too small, the detection accuracy will be low; if you choose a large number of APIs will cause redundancy of data, reduce detection efficiency, and increase detection time complexity. Through the final analysis of the experimental results, the top 20 APIs with the highest correlation coefficients with malicious applications were selected. The ranking of sensitivity calculation results is shown as Table 1.

Table 1. The 20 most sensitive API calls

Number	Score	API calls
1	0.467	sendMultipartTextMessage ( )
2	0.426	getNETWORKCountryIso ( )
3	0.402	openConnection( )
4	0.385	chmod ( )
5	0.343	abortBroadcast ( )
6	0.301	writeTextMessage ( )
7	0.279	writeExternalStorageState ( )
8	0.266	sendTextMessage ( )
9	0.257	getLine1Number ( )
10	0.252	getLastKnownLocation ( )
11	0.209	getSimOperator ( )
12	0.198	getAccountsByType ( )
13	0.194	getDisplayMessageBody ( )
14	0.193	com.android.contacts ( )
15	0.188	getOutputStream ( )
16	0.173	getDeviceId ( )
17	0.165	getInputStream ( )
18	0.161	startService ( )
19	0.157	getRunningTasks ( )
20	0.153	updateConfigurationLocked ( )

Generation of Eigenvectors. We create a 20-dimensional feature vector $[APIs]_{1*20}$ for each experimental sample, and unified the format of the feature vector as: $[X_1 : Y_1; X_2 : Y_2; X_3 : Y_3; \cdots X_i : Y_i; \cdots X_{20} : Y_{20}]$ (i = 1,2,3... 20). The X_i represents the i-th API calls in Table 1, and Y_i represents whether this API calls exists in the sample. If it exists, it is set to 1, otherwise it is set to 0. Then input the generated feature vector into kNN classifier, DT classifier and SVM classifier respectively. In this paper, we use 450 malicious samples and 450 benign samples to train, and then use 100 samples to detect.

4 Detection System Design

The malware detection flowchart is shown in Fig. 5. This section is the detection phase, which mainly uses the integrated learning model to detect unknown applications.

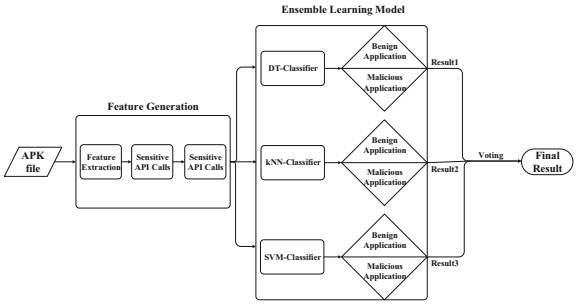

Fig. 5. Flow chart of malware detection

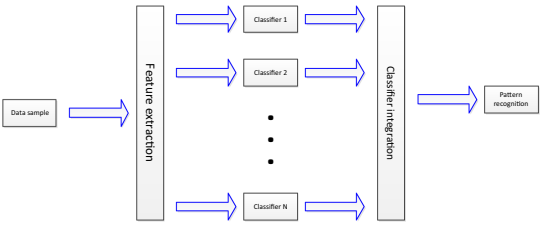

Fig. 6. Multi-classifier integrated learning model implementation diagram

4.1 Ensemble Learning Detection Model

Classification is a kind of supervised learning by training a classifier in the samples of the known category so that it can classify the unknown samples, as shown in Fig. 6. However, the detection accuracy of a single classification algorithm is not high, and it has a certain randomness. Therefore, this paper adopts the set learning method [12]. In this paper, we use DT, kNN and SVM as the basic classifier to classify the samples.

4.2 Base Classifiers

In this paper, kNN classifier, DT classifier and SVM classifier are used as the basic classifier of the integrated learning model.

KNN Classifier. The kNN algorithm is a classification algorithm. The idea of the algorithm is: A sample is most similar to the k samples in the data set. If most of the k samples belong to a certain category, the sample also belongs to this category. The choice of k value in this algorithm is critical. Too small k value will cause the model to be complicated and prone to over-fitting. Too large k value will reduce the accuracy of prediction. Sometimes benign and malicious applications call the same API, resulting in overlapping sample sets. For this, the classification performance of the kNN classifier is more advantageous than other classifiers.

In summary, the selection of k value is very important in the kNN algorithm. As shown in Fig. 7, according to the test data results, when the k value is 5, the classification accuracy of the kNN algorithm is the highest. Therefore, the value of k is 5 in this paper.

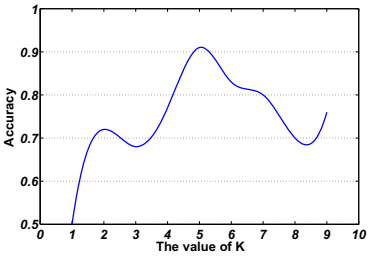

Fig. 7. The accuracy of different k value on detection results

DT Classifier. The algorithm of classification decision tree learning is a process of recursively selecting the optimal feature, and segmenting the training data according to the feature, so that each sub-data set has the best classification process. For the DT classifiers, the "information gain" is similar to the MI model used in feature extraction. Therefore, the DT classifier is very suitable as the basic classifier in the ensemble learning model.

SVM Classifier. Support vector machines are a two-class classification model (or called classifier). Its classification idea is to solve the separation hyperplane that can correctly divide the training data set and have the largest geometric interval for the sample set of positive and negative examples. Since the SVM algorithm was originally designed for binary classification problems, it has unique advantages and better performance in application scenarios to determine application maliciousness. This not only helps us to pay more attention to key samples and eliminate a large number of redundant samples, but also destined to have better robustness of SVM algorithms. Because of the advantages and good classification performance of the SVM algorithm, we also use the SVM classifier

as a set of base classifiers in the experiments designed in this paper. Together with the kNN classifier and the DT classifier mentioned above, it forms the base classifier for integrated learning.

4.3 Weighting Combination Strategy

In this paper, we use [apkName, result-DT], [apkName, result-kNN] and [apkName, result-SVM] represent the detection results given by the DT classifier, kNN classifier and SVM classifier respectively. If the detected application is malware, result-DT, result-kNN, and result-SVM set the value to 0. Otherwise it will be set to 1.

The reason why the detection threshold is 0.5 is that the probability of the detection result of the detector on the sample is equal. In detail, we use the linear weighted sum method (LWSM) to calculate the final result. The linear weighted sum is calculated as formula (4):

$$Final - Result = P_1 * X_1 + P_2 * X_2 + \cdots + P_n * X_n = \sum (P_i * X_i) \quad (4)$$

Based on this, the possible detection results are shown in the following Table 2:

Table 2. Detection results

Result	Classifier		
	DT	kNN	SVM
Benign	1	1	1
Benign	1	0	1
Benign	0	1	1
To be identified	0	0	1
To be identified	1	1	0
Malware	0	1	0
Malware	1	0	0
Malware	0	0	0

If the final result is greater than 0.5, the application will be classified as a benign application. If the final result is less than 0.5, the application is classified as a malicious application. If the result is equal to 0.5, then further manual recognition is required. However, the experimental results show that the probability of this situation is low, about 2–4%.

5 The Results of Experimental

5.1 Experimental Environment

In this experiment,we selected 1044 experimental samples as data sets. Among them, 528 malicious applications are download from the malicious sample set of the virusShare.com [13].And 516 normal Android applications are provided from third-party application market and Google Android Market [14] by using web crawler programs.The experimental environment is: operating system of Windows 10, processor: Intel Core i7, 16 GB of memory, Python 2.7 scripting languages.

5.2 Evaluation Indeces

Classification performance is commonly assessed by the accuracy, true positive rate (TPR), false positive rate (FPR) and precision.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$TPR = \frac{TP}{TP + FN} \tag{6}$$

$$FPR = \frac{FP}{FP + TN} \tag{7}$$

$$Precision = \frac{TP}{FP + TP} \tag{8}$$

Here, TP and TN are the numbers of true and false samples that are correctly labeled by the classifier, respectively, and FN and FP are the numbers of true and false samplesthat are incorrectly labeled by the classifier, respectively.

5.3 Analysis of Experimental

In order to verify that the detection method proposed in this paper has good applicability, we conducted multiple classification tests on a large number of samples.

Detection Model Performance. This paper compares the experimental results of whether to use sensitive APIs. The results are shown in Fig. 8.

In Fig. 8, when a set of sensitive API calls are used, the effect of the detection model is significantly improved in terms of accuracy, TPR and accuracy. Among them, accuracy can reach 93% while the precision can reach 95%. In addition, our method proposed in this paper can achieve a TPR of 89%. All in all, our malicious application detection method based on sensitive API has better detection results.

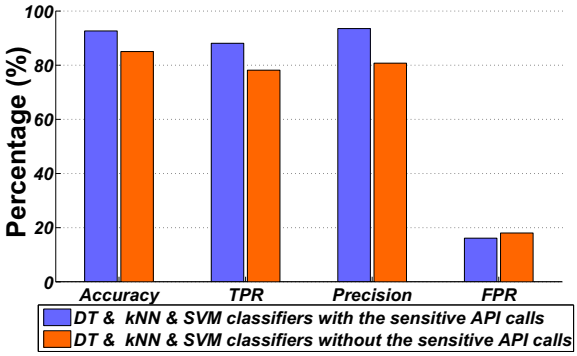

Fig. 8. Detection model performance

The Effect of Detection Model. Figure 9 and Fig. 10 respectively show that our ensemble learing model has a higher detection rate and a lower FPR in the six comparative experiments conducted. Among them, it can achieve the optimal accuracy of 94.8% and the average detection accuracy above 94%.

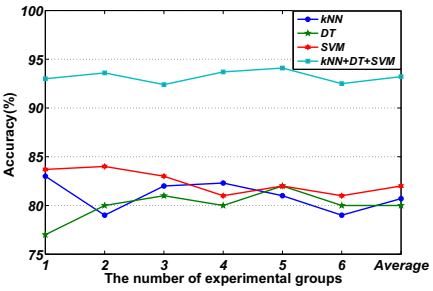

Fig. 9. Contrast of accuracy with different classifiers

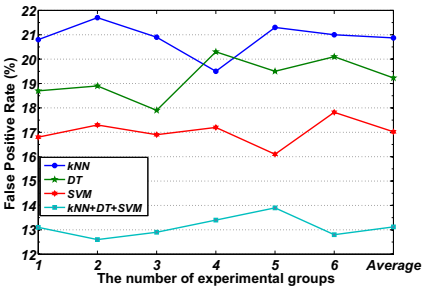

Fig. 10. Contrast of FPR with different classifiers

The Comparison of Different Classifiers. As shown in Fig. 11, we have selected five different classification algorithms to analyze our experimental results. It can be seen that combining the results of TPR and PRE, when the integrated learning model of DT + kNN + SVM algorithm is used, the entire system reaches the best performance. Therefore, they are used as basic classifiers.

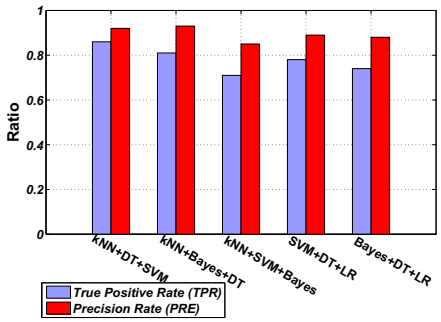

Fig. 11. The effect of different classifiers

The Comparison of Different Weights. We also conducted corresponding experiments on the different weights occupied by the kNN classifier, DT classifier and SVM classifier in the integrated learning module, and gave the classification results of the DT, kNN and SVM classifiers, respectively. Different weights, and the final test result is calculated according to the linear weighted sum. It can be seen from the calculation that there are different weight distribution combinations in total. In this paper, we conducted corresponding experiments on each combination of weights. As shown in Fig. 12, we selected nine representative weight combinations for drawing. It can be seen that the detection result of the whole system is the most optimal when the weight of the DT classifier is 0.2, the weight of kNN is 0.3 and the weight of SVM classifier is 0.5.

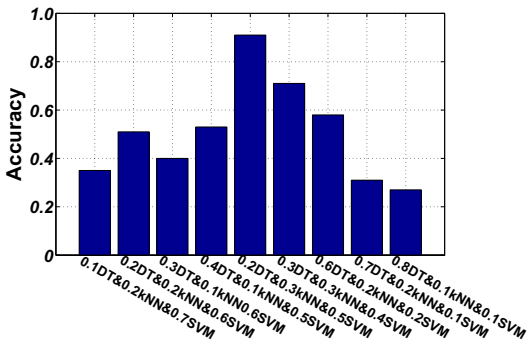

Fig. 12. The effect of different weights

6 Conclusion

This paper proposes an Android malicious application detection model based on integrated learning. By extracting API functions called by Android applications and combining MI models to generate a set of sensitive APIs. Then select the top

20 sensitive API functions as the feature library to generate 20-dimensional feature vectors. The integrated learning model based on kNN classifier, DT classifier and SVM classifier is used to effectively detect unknown Android applications. Experimental results show that this method has achieved good results in Precision, TPR and Accuracy. However, the method proposed in this paper has a high FPR due to the detection value of 0.5. In future research, we will conduct more experimental studies to reduce the FPR on the basis of ensuring the accuracy and accuracy of detection.

References

1. IDC new report: Android account for 87% market share in 2019, iPhone only accounts for 13%. https://baijiahao.baidu.com/s?id=1644272367710328052&wfr=spider&for=pc. Accessed 10 Sept 2019
2. Android. https://www.android.com/. Accessed 30 Nov 2016
3. Enck, W., Gilbert, P., Han, S., et al.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 5 (2014)
4. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (2011), pp. 15–26. ACM (2011). https://doi.org/10.1145/2046614.2046619
5. Wu, S., Wang, P., Li, X., Zhang, Y.: Effective detection of android malware based on the usage of data flow APIs and machine learning. Inf. Softw. Technol. **75**, 17–25 (2016)
6. Onwuzurike, L., Mariconti, E., Andriotis, P., De Cristofaro, E., Ross, G., Stringhini, G.: MaMaDroid: detecting android malware by building Markov chains of behavioral models (extended version). ACM Trans. Priv. Secur. **22**(2), 1–34 (2019)
7. Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. IEEE Trans. Infor. Forensics and Secur. **14**(3), 773–788 (2019). https://doi.org/10.1109/TIFS.2018.2866-319
8. Zhao, C., Zheng, W., Gong, L., et al.: Quick and accurate android malware detection based on sensitive APIs. In: IEEE International Conference on Smart Internet of Things. IEEE Computer Society, pp. 143–148 (2018)
9. Felt, A.P., Chin, E., Hanna, S., et al.: Android permissions demystied. In: ACM Conference on Computer & Communications Security, vol. 10, p. 627 (2011)
10. Guyon, I., Elisseeff, A., et al.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**(6), 1157–1182 (2013)
11. Wang, X., Feng, D., Liu, J.: Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
12. Xiang, C., Yang, P., Tian, C., Liu, Y.: Calibrate without calibrating: an iterative approach in participatory sensing network. IEEE Trans. Parallel Distrib. Syst. **26**(2), 351–356 (2015)
13. Virusshare. http://virusshare.com. Accessed 30 Sept 2017
14. Google android market. https://play.google.com/store/apps?feature=corpusselector. Accessed 30 Jan 2017