Comparative Analysis of Ensemble Methods for Classification of Android Malicious Applications

Meghna Dhalaria, Ekta Gandotra(✉), and Suman Saha

Department of Computer Science and Engineering,
Jaypee University of Information Technology, Waknaghat, Solan, H.P., India
meghna8aug@gmail.com, ekta.gandotra@gmail.com,
suman.saha@juit.ac.in

Abstract. Currently, Android smartphone operating systems are the most popular entity found in the market. It is open source software which allows developers to take complete benefit of the mobile operation device, but additionally increases sizable issues related to malicious applications. With the increase in Android phone users, the risk of Android malware is increasing. This paper compares the basic machine learning algorithms and different ensemble methods for classifying Android malicious applications. Various machine learning algorithms such as Random Forest, Logistic Regression, Support Vector Machine, K-Nearest Neighbor, Decision Tree and Naive Bayes and ensemble methods like Bagging, Boosting and Stacking are applied on a dataset comprising of permissions, intents, Application programming interface (API) calls and command signatures extracted from Android applications. The results revealed that the stacking ensemble techniques performed better as compared to the Bagging, Boosting and base classifiers.

Keywords: Android malware classification · Machine learning · Ensemble techniques · Bagging · Boosting · Stacking

1 Introduction

Android is the widely used mobile platform across the world with more than 85% of the market share [1]. The increasing use of Android-based applications (apps) is causing the growth of malware. Nearly 17.5 million Android users downloaded malicious applications from the official Google Play Store in 2017 [2]. These malicious apps create several serious threats such as information leakage, system damage and financial loss etc. According to the McAfee report [3], the growth of Android malware is increasing rapidly with approx 750 million in 2018. Android malware may be embedded in various applications such as gaming, educational and banking apps etc. These infected applications can compromise privacy and security by permitting unauthorized access to rooting devices, private sensitive information, etc. Earlier, most of the malware detection methods were based on signature-based approach. It uses a database of known malware signatures and compares each application against this database. The drawback of this method is that it is not suitable for detection of new

M. Singh et al. (Eds.): ICACDS 2019, CCIS 1045, pp. 370–380, 2019.
https://doi.org/10.1007/978-981-13-9939-8_33

malware (i.e. zero-day malware) whose signatures do not exist in the database. Recently, researchers started to use machine learning methods for malware classification. The problem in machine learning technique is that it gives high false positive and false negative rate [4]. Ensemble techniques such as Bagging, Boosting and Stacking are applied in order to improve accuracy. The goal of this paper is to compare the different ensemble methods for Android malware classification. The paper is organized as follows: Sect. 2 provides an overview of related work. Section 3 presents the approach used. Section 4 provides the experimental results and their comparative analysis. Finally, Sect. 5 presents the conclusion.

2 Literature Review

In this section, numerous contributions have been explored with the aid of the researchers in the field of Android malware detection using machine learning. Zhou and Jiang [5] characterized existing Android malware from diverse components, including the permissions requested. They recognized the permissions which are extensively asked in both benign and malicious apps. The author found that malicious apps have a tendency to request extra permissions than benign ones. In 2012, Sanz et al. [6] introduced a new technique to detect Android malware applications through machine learning strategies with the aid of analyzing the extracted permissions from the application itself. In [7], the author presented a method MAMA that extracts numerous features from the Android manifest to build classifiers and detect malware. Huang et al. [8] presented a technique for detection of Android malicious applications based on 20 features. Their experimental results show that an individual classifier is able to detect about 81% of malware applications. In [9], the author developed a tool named Marvin that creates a risk by examining an application using static and dynamic features. In 2015, Bhandari et al. [10] developed an approach DRACO that combines both static and dynamic analysis. It explains the features that contributing to the maliciousness of the examined application and generates the score. In [11], the author introduced SigPID named as Significant Permission Identification for detection of Android malware. The detection device based totally on permission usage to deal with rapid growth in Android malware. SigPID used machine learning based classification method such as SVM and Decision tree to classify the apps into malware or benign. The results show that SVM achieves 90% of recall, precision, F-measure and accuracy. In 2015, Cen et al. [12] proposed a malware detection method primarily based on permissions and API calls. They applied probabilistic discriminative model based on RLR (Regularized Logistic Regression) and compared with other classifier named as K-NN, decision tree, SVM and Naive Bayes. Yerima et al. [13] proposed a novel classifier fusion approach named as DroidFusion which is based on the multilevel architecture that enables the combination of algorithms for improving the accuracy. They applied the various ranking algorithm on their predictive accuracy in order to drive final classifier. Their experimental results show that the fusion method performs better for improving accuracy than the ensemble learning algorithm. Wang et al. [14] applied a different machine learning algorithm named as SVM, Random Forest and Logistic regression with static analysis for detection of Android malware apps. For training machine

learning algorithms they used platform-specific static features and app specific static features. Experimental results demonstrate that logistic regression performs better in comparison to other classifiers with 96% of TPR (True Positive Rate) and 0.06% of FPR (False Positive Rate). In [15], the author introduced a novel dynamic evaluation framework, referred to as EnDroid, which automatically extracts multiple varieties of dynamic features to implement effective malware detection. For effective detection of malware, they applied the stacking ensemble technique. Their experimental outcomes show that stacking perform better for the detection of Android malware. This paper presents a comparative analysis of base classifiers and ensemble methods for classification of Android applications.

3 Methodology Used

This section discusses the approach followed for comparing the machine learning algorithms and ensemble methods for detecting and classifying Android applications into malicious and benign. First of all, a dataset [5] comprising of permissions, intents, API calls and command signatures extracted from Android malicious and benign applications is downloaded. Six machine learning algorithms i.e. Naive Bayes (NB), Random Forest (RF), Logistic Regression (LR), K-Nearest Neighbor (K-NN), Decision Tree (DT) and Support Vector Machine (SVM) and three ensemble technique i.e. Bagging, Boosting and Stacking are applied on the dataset using WEKA (Waikato Environment for Knowledge Analysis) [16] library and their performance is evaluated based on different parameters. For stacking ensemble technique, the topmost four classifiers (on the basis of accuracy) are combined in the group of three making four different combinations. The LR is used as level-2 meta-classifier in stacking. Afterward, comparative analysis is carried out on the results obtained. The details of the dataset used, machine learning and ensemble algorithms used are given in the following sub-sections. Figure 1 depicts the methodology of the proposed work.

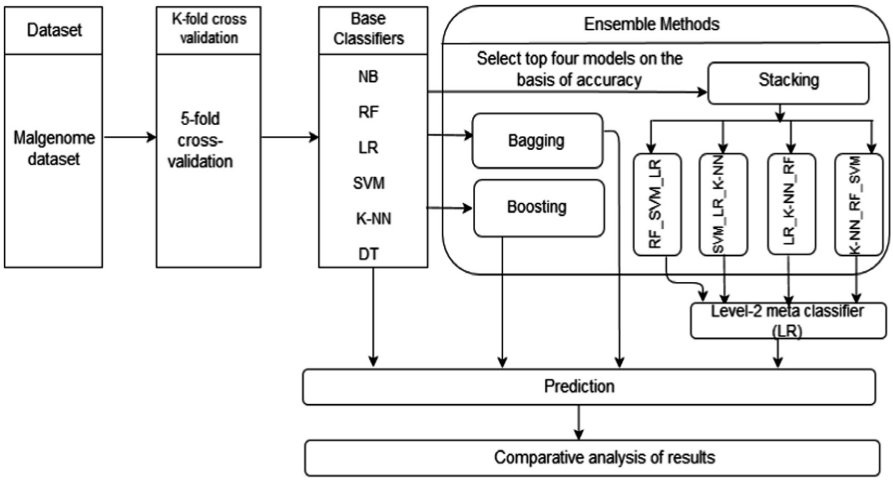

Fig. 1. Workflow of methodology used

Dataset Used
Malgenome [5] dataset is used for this work. It consists of 3800 number of instances which includes 1260 malware and 2539 benign. There are total 215 attributes which are categorized into 4 parts i.e. API Call Signature, Manifest Permission, Command Signature, and Intent. At the time of installation, the permission is granted by the users and these are declared in Android-Manifest file. API calls are needed to interact with the device. Intents are also described in the Android Manifest file. It is the conceptual information about an operation, with which we can infer the intentions of the apps.

Base Classifiers

Decision Tree
The DT [17] is used for building regression and classification model in the form of a tree structure. It focuses on an easily understandable representation form and is one of the most common learning methods. It can easily be visualized in tree structure format. A decision tree is built by iteratively splitting the dataset on the attribute that separates the data into the different existing classes until a stopping criterion is reached.

Random Forest
RF [18] is an ensemble learning method that creates numerous regression trees and aggregates their results. It trains every tree independently by the usage of a random sample of the data. This randomness allows making the model more powerful. The random forest model is excellent at managing tabular records with numerical functions or categorical functions with fewer than loads of categories.

Support Vector Machine
The SVM [19] algorithm, uses hyperplane to divide the n-dimensional space data into two regions. It calculates the maximal margin between all dimensions i.e. it is creating the largest distance between instances, which is reducing the generalization Error. The basic approach to classify the data starts by trying to create a function that splits the data points into the corresponding labels with (a) the least possible amount of errors or (b) with the largest possible margin.

Naive Bayes
NB [20] is a classification technique based on Bayes' theorem. This classifier is widely using in text estimation. For instance, many spam filters are using it in order to divide acceptable content from unacceptable. Usually, the accuracy of this method is relatively low in contrast with other approaches. However, an advantage of this technique is a very high speed of classification and also a very good level of tolerance to missing values. Additionally, NB algorithm characterized by low tolerance to redundant attributes. Continuous features are not permitted here.

K-Nearest Neighbor
K-NN [21] model is also a type of supervised learning algorithm. It is the simplest and easy than other machine learning techniques. This algorithm is representative of lazy algorithms. It is based on the assumption that records within a dataset are generally having the same properties. K-NN algorithm is relatively slow in the classification of

new instances coming into the model but fast during the training process. Also, this algorithm is very sensitive to noise in the dataset.

Ensemble Techniques

Bagging: Bagging is also known as **B**ootstrap **Agg**regat**ing**. Multiple models of the same learning algorithm are generated over a subset of the training dataset using random sampling with replacement. For combining the models, the two methods are used i.e. majority voting and averaging. In majority voting, the final prediction is done on the basis of votes of each classifier. In averaging, it takes an average of the predictions of each classifier. In our work, we have used the majority vote method. Bagging helps to avoid the problem of overfitting and can reduce variance [22].

Bagging Algorithm

For training:

1. Repeat step 2 to 3 for each iteration i = 1, 2, …, n
2. Create bootstrap samples of the training dataset using random sampling with replacement.
3. Train different classifiers on these samples (NB, K-NN, DT etc.)

For testing:

1. Use a new dataset, to make predictions using base classifiers.
2. Combine the results of all models on the basis of majority voting.

Boosting: This method is used for improving the predictions of the model. Boosting technique selects instances which give the wrong prediction and modify the weights. Boosting is a little variation on bagging. In boosting, firstly equal weights are assigned to all instances. Train the classifiers to make predictions of wrongly classified instances then modify the weights of incorrectly predicted instances. In the end, take the weighted mean of all weak learners to make a strong learner i.e. final model [23]. There are different boosting algorithms such as AdaBoost, Gradient Tree Boosting and XGBoost. The AdaBoost (**Ada**ptive **Boost**ing) algorithm is used to perform boosting in our work.

Boosting Algorithm

1. Assign equal weights to all instances.
2. Train the classifier to make predictions.
3. Assign higher weights to wrongly classified instances.
4. Repeat step 2 & 3 till the classifier correctly predict the instances.

Stacking: Stacking is also known as stacked generalization. It deals with combining multiple classifiers generated by different machine learning algorithms. The process of stacking can be divided into two phases: In the first phase, all the algorithms are trained using the training data. In the second phase, the predictions from multiple models are used as input to the second level to build a new model. This model is used for the prediction on test data [24].

Stacking Algorithm

1. Select topmost four classifiers (on the basis of accuracy) combine these in the group of three making four different combinations.
2. Train these models using a complete training dataset.
3. Construct a new dataset of predictions made from multiple base-level classifiers.
4. Train a meta-model i.e. LR using the new dataset created in step 3.
5. Make predictions using this newly formed model.

4 Experimental Results

This section discusses the experimental results obtained. Six different classifiers i.e. NB, LR, RF, K-NN, DT and SVM (explained in Sect. 3) are executed on WEKA 3.8 under Intel Core i3 processor, 64 bit, 2 GB RAM. All the classification models and ensemble methods are trained using 5-fold cross-validation. The parameters used for evaluating various models are True Positive Rate (TPR), False Positive Rate, Precision, F-measure and Accuracy.

Table 1. Performance evaluation of base classifiers

Classifier	TPR	FPR	Precision (%)	F-measure (%)	Accuracy (%)
NB	0.959	0.044	95.9	95.9	95.8
RF	0.991	0.016	99.1	99.1	99.0
LR	0.974	0.026	97.4	97.4	97.3
SVM	0.990	0.011	99.0	99.0	99.0
K-NN	0.986	0.015	98.6	98.6	98.5
DT	0.970	0.037	97.0	97.0	96.9

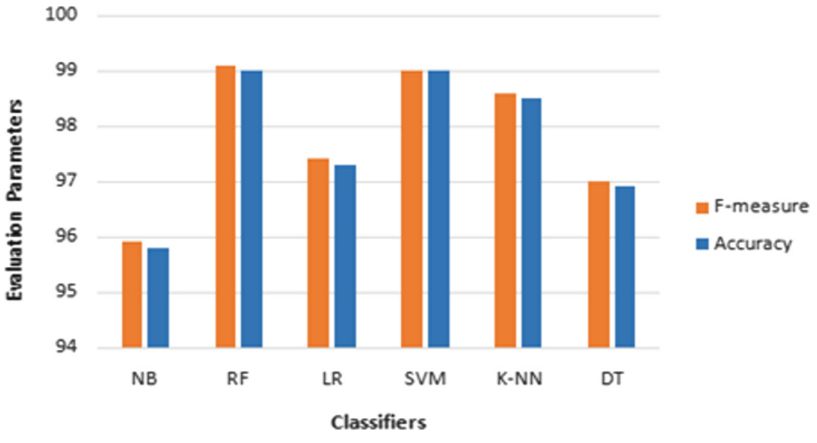

Fig. 2. Comparative analysis of base classifiers

Table 1 represents the result of six base classifiers. The results show that SVM and RF give the best accuracy i.e. 99% followed by K-NN which gives an accuracy of 98.5%. The TPR of RF and SVM is also high i.e. 0.99 and FPR is low i.e. 0.016 (RF) and 0.011 (SVM) respectively.

Figure 2 represents the comparison of six base classifiers on the basis of F-measure and Accuracy. Herein SVM and RF classifiers are proved to be the best among used base classifiers for the Malgenome dataset.

Table 2. Performance evaluation of classifiers using bagging ensemble technique

Classifier	TPR	FPR	Precision (%)	F-measure (%)	Accuracy (%)
NB_Bagging	0.961	0.042	96.1	96.1	96.0
RF_Bagging	0.992	0.016	99.2	99.2	99.1
LR_Bagging	0.979	0.021	97.9	97.9	97.9
SVM_Bagging	0.989	0.011	99.0	98.9	98.9
K-NN_Bagging	0.988	0.015	98.8	98.8	98.8
DT_Bagging	0.985	0.021	98.5	98.5	98.4

Table 2 represents the performance of six different classifiers using Bagging ensemble technique. The results show that classifiers with bagging namely NB, LR, RF, K-NN and DT give better results as compared to base classifiers. There is no improvement in results using the bagging technique in case of SVM because bagging work well with unstable classifiers. There is a minor improvement in TPR and FPR for classifiers with Bagging as compared to the base classifiers.

Table 3 depicts the performance of six classifiers using Boosting ensemble technique. The results show that all the classifiers with boosting except SVM give better results as compared to base classifiers. It is found that classifiers with AdaBoost algorithm have performed better than classifier with bagging ensemble technique with majority voting. There is an improvement in TPR and FPR for classifiers with AdaBoost algorithm as compared to the classifiers with bagging.

Table 3. Performance evaluation of classifiers using boosting ensemble technique

Classifier	TPR	FPR	Precision (%)	F-measure (%)	Accuracy (%)
NB_Boosting	0.977	0.030	97.7	97.7	97.7
RF_Boosting	0.992	0.013	99.2	99.2	99.2
LR_Boosting	0.974	0.026	97.4	97.4	97.3
SVM_Boosting	0.988	0.014	98.8	98.8	98.8
K-NN_Boosting	0.986	0.015	98.6	98.6	98.6
DT_Boosting	0.991	0.012	99.1	99.1	99.0

Figure 3 represents the comparison of the base classifiers with bagging and boosting ensemble technique on the basis of F-measure. Both Bagging and Boosting showing better accuracy than base classifiers. Adaboost with NB and DT are performing better than their bagged versions. Bagging with LR and K-NN has better accuracy than their boosted version. Bagging and Boosting with RF is giving almost same result.

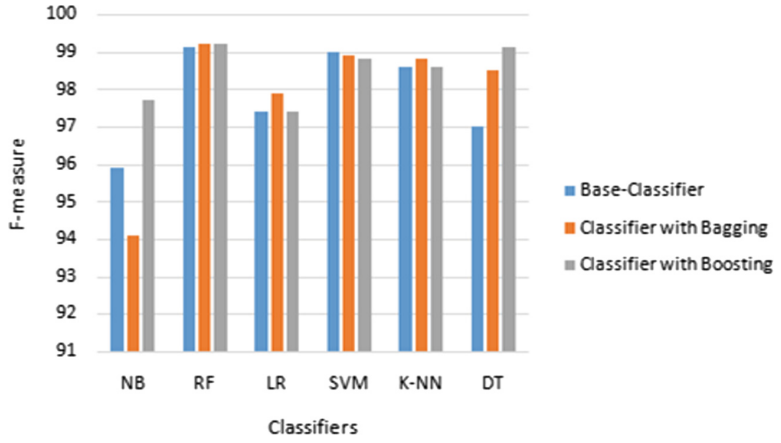

Fig. 3. Comparison of the base classifiers with bagging and boosting ensemble techniques on the basis of F-measure

Herein classifier with boosting i.e. NB, RF and DT are the best among other classifiers. The classifier with bagging is also performing better with some classifiers i.e. RF, LR and K-NN.

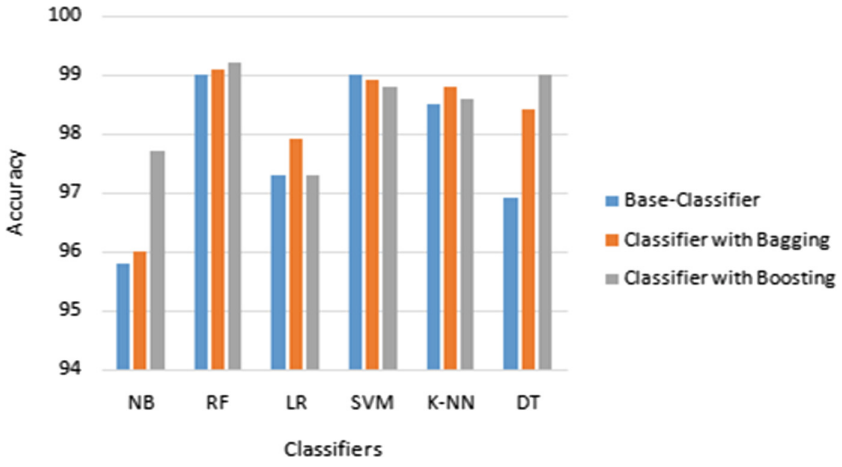

Fig. 4. Comparison of the base classifiers with bagging and boosting ensemble techniques on the basis of Accuracy

Figure 4 represents the comparison of the base classifier with bagging and boosting ensemble technique on the basis of Accuracy. Herein classifier with boosting namely NB, RF and DT are shown to be the best among other classifiers. The classifier with bagging is also performing better with some classifiers i.e. LR and K-NN.

For applying stacking ensemble method, we ranked the base classifiers on the basis of their accuracy. The ascending order of top four base classifiers is LR, K-NN, RF and SVM. Following four stacked ensemble models (each consisting of three base classifiers out of the top four) are designed and tested on the Malgenome dataset using 5-fold cross-validation.

M1 (RF_SVM_LR)
M2 (SVM_LR_K-NN)
M3 (LR_K-NN_RF)
M4 (K-NN_RF_SVM)

Table 4. Performance evaluation of four stacking ensemble models

Ensemble models	TPR	FPR	Precision (%)	F-measure (%)	Accuracy (%)
RF_SVM_LR	0.993	0.008	99.3	99.3	99.3
SVM_LR_KNN	0.992	0.011	99.2	99.2	99.1
LR_KNN_RF	0.993	0.009	99.3	99.3	99.2
KNN_RF_SVM	0.993	0.009	99.3	99.3	99.2

Stacking ensemble method uses a stack of classifiers in order to achieve better results as compared to the individual classifier. The Logistic regression is used as level-2 meta-classifier. Table 4 represents the performance of four ensemble models. The results show that almost all the four models designed using stacking ensemble techniques perform better than the base classifiers and other ensemble methods i.e. Bagging and Boosting. The accuracy obtained by RF_SVM_LR is 99.3% followed by LR_K-NN_RF and K-NN_RF_SVM which provide an accuracy of 99.2%.

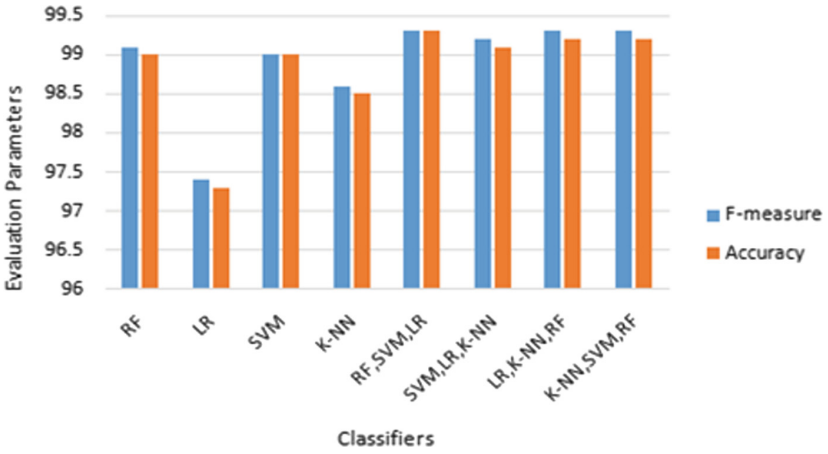

Fig. 5. Comparison of top four base models and stacking ensemble models

Figure 5 depicts the comparative analysis of the top four base models with the stacking ensemble approach in terms of F-measure and Accuracy. The stacking ensemble approach gives the highest accuracy to classify malicious apps. It is clear from the results that the models designed with stacking ensemble technique show significant improvement over base classifiers.

5 Conclusion

With the increase in Android phone users, the risk of Android malware is increasing. So there is a need to develop an effective technique for better classifying the malware. This paper presented a comparative analysis of base classifiers and three ensemble techniques i.e. Bagging, Boosting and stacking for classifying Android apps. It is concluded that ensemble techniques perform better as compared to the base classifiers. Further stacking ensemble technique outperforms in comparison to Bagging and Boosting ensemble technique. The comparison is done on the basis of F-measure and Accuracy. It is revealed from the results that the overall stacking ensemble model has improved accuracy in contrast to the base classifier accuracy. The result shows that the combination of RF_SVM_LR performs better as compared to the other ensemble models. The accuracy obtained is 99.3%.

References

1. Canalys: Over 1 billion Android-based smart phones to shipin 2017
2. Arora, A., Garg, S., Peddoju, S.K.: Malware detection using network traffic analysis in Android based mobile devices. In: 8th IEEE International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 66–71 (2014)
3. McAfee Labs Threats Report: March 2018 Visit: www.mcafee.com/March2018ThreatsReport
4. Gandotra, E., Bansal, D., Sofat, S.: Malware analysis and classification: a survey. J. Inf. Secur. **5**, 56 (2014)
5. Jiang, X., Zhou, Y.: Dissecting Android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109. IEEE (2012)
6. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: PUMA: permission usage to detect malware in Android. In: Herrero, Á., et al. (eds.) International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions. AISC, vol. 189, pp. 289–298. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33018-6_30
7. Sanz, B., et al.: MAMA: manifest analysis for malware detection in Android. Cybern. Syst. **44**, 469–488 (2013)
8. Huang, C.Y., Tsai, Y.T., Hsu, C.H.: Performance evaluation on permission-based detection for Android malware. In: Pan, J.S., Yang, C.N., Lin, C.C. (eds.) Advances in Intelligent Systems and Applications - Volume 2. SIST, vol. 21, pp. 111–120. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35473-1_12
9. Lindorfer, M., Neugschwandtner, M., Platzer, C.: Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 2, pp. 422–433. IEEE (2015)

10. Bhandari, S., Gupta, R., Laxmi, V., Gaur, M.S., Zemmari, A., Anikeev, M.: DRACO: DRoid Analyst COmbo an Android malware analysis framework. In: Proceedings of the 8th International Conference on Security of Information and Networks, pp. 283–289. ACM (2015)
11. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., Ye, H.: Significant permission identification for machine-learning-based Android malware detection. IEEE Trans. Industr. Inf. **14**, 3216–3225 (2018)
12. Cen, L., Gates, C.S., Si, L., Li, N.: A probabilistic discriminative model for Android malware detection with decompiled source code. IEEE Trans. Dependable Secure Comput. **12**, 400–412 (2015)
13. Yerima, S.Y., Sezer, S.: DroidFusion: a novel multilevel classifier fusion approach for Android malware detection. IEEE Trans. Cybern. **99**, 1–14 (2018)
14. Wang, X., Wang, W., He, Y., Liu, J., Han, Z., Zhang, X.: Characterizing Android apps' behavior for effective detection of malapps at large scale. Future Gener. Comput. Syst. **75**, 30–45 (2017)
15. Feng, P., Ma, J., Sun, C., Xu, X., Ma, Y.: A novel dynamic Android malware detection system with ensemble learning. IEEE Access **6**, 30996–31011 (2018)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explorations Newsl **11**(1), 10–18 (2009)
17. Quinlan, J.R.: Induction of decision trees. Mach. Learn. 81–106 (1986)
18. Liaw, A., Wiener, M.: Classification and regression by randomForest. R News **2**, 18–22 (2002)
19. Keerthi, S.S., Gilbert, E.G.: Convergence of a generalized SMO algorithm for SVM classifier design. Mach. Learn. **46**, 351–360 (2002)
20. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. Mach. Learn. **29**, 103–130 (1997)
21. Shakhnarovish, G., Darrell, T., Indyk, P. (eds.): Nearest-Neighbor Methods in Learning and Vision. MIT Press, Cambridge (2005)
22. Breiman, L.: Bagging predictors. Mach. Learn. **24**, 123–140 (1996)
23. Quinlan, J.R.: Bagging, boosting, and C4.5. In: AAAI/IAAI, vol. 1, pp. 725–730 (1996)
24. Ting, K.M., Witten, I.H.: Issues in stacked generalization. J. Artif. Intell. Res. **10**, 271–289 (1999)