Towards a Mobile Malware Detection Framework with the Support of Machine Learning

Dimitris Geneiatakis, Gianmarco Baldini(✉), Igor Nai Fovino, and Ioannis Vakalis

European Commission, Joint Research Centre (JRC), Cyber and Digital Citizens' Security Unit, Via Enrico Fermi 2749, 21027 Ispra, Italy
gianmarco.baldini@ec.europa.eu

Abstract. Several policies initiatives around the digital economy stress on one side the centrality of smartphones and mobile applications, and on the other call for attention on the threats to which this ecosystem is exposed to. Lately, a plethora of related works rely on machine learning algorithms to classify whether an application is malware or not, using data that can be extracted from the application itself with high accuracy. However, different parameters can influence machine learning effectiveness. Thus, in this paper we focus on validating the efficiency of such approaches in detecting malware for Android platform, and identifying the optimal characteristics that should be consolidated in any similar approach. To do so, we built a machine learning solution based on features that can be extracted by static analysis of any Android application, such as activities, services, broadcasts, receivers, intent categories, APIs, and permissions. The extracted features are analyzed using statistical analysis and machine learning algorithms. The performance of different sets of features are investigated and compared. The analysis shows that under an optimal configuration an accuracy up to 97% can be obtained.

1 Introduction

Digital Single Market (DSM) strategy[1], and other policy initiatives recognize the potentialities of digital business for innovation and growth. Smarthphones and mobile applications are considered a basic component in this ecosystem. This is not only because lately the 50% of web page traffic is generated by mobile devices as reported in [1], but also mobile networks enable users to interact with digital services at any time and almost at any place. Thus, service providers offer to users both web based applications and their mobile counterpart.

However, to exploit DSM capacities in mobile environments various challenges should be faced. Among them security, privacy and trust is of high concern. This is of especially importance for mobile applications that users can install either from trusted official sites (*i.e.,* Google Play) or through third

[1] https://ec.europa.eu/commission/priorities/digital-single-market_en.

E. Gelenbe et al. (Eds.): Euro-CYBERSEC 2018, CCIS 821, pp. 119–129, 2018.
https://doi.org/10.1007/978-3-319-95189-8_11

party applications stores. Note that even if mobile applications are inspected (*e.g.,* through Google bouncer), for security flaws before publishing, it is rather impossible to be completely sure about the quality of any given application from a security and privacy perspective as adversaries always find new ways and techniques to bypass the underlying protection mechanisms. Furthermore, third parties stores do not follow the same security policies as Google play and hence is most probable to be used as a vehicle for malware distribution.

However, lately systems' security have been highly enhanced by introducing different solutions [2,3] either at operating system or at application layer works, malware detection as a part of it is still an on-going and challenging research problem. Various research works [4,5] have shown that not only malicious software (malware) can get access to private information but also goodware might try to invade to users' digital space as it is considered the main asset for digital business in this new era, and consequently can threaten their private sphere. So a major question, considering the vast number of available mobile applications (apps), is whether users can be informed if a given app can act maliciously to a certain degree. To do so it is of high importance to:

1. understand mobile applications characteristics that can be used for such a classification
2. provide a solution capable of analyzing the large scale landscape and learn to identify potential problems and unknown patterns in the fly.

In this context, anomaly detection solutions with emphasis on machine learning (ML) have been considered as an alternative option, to traditional ones, recently. This is because (a) humans are incapable of identifying (common) patterns in a high frequency data, (b) cyber domain is currently supported by big data meaning that high volume of data can be used for developing the appropriate base line behaviours, and (c) ML assisted tools the last decades have been used successfully in diverse domains such as text processing, health-care, finance, *etc.*

We believe that a well-defined model relying on the advantages of ML can be a promising ally for improving and enhancing the current level of cyber security protection/identification solutions. Currently, related works *e.g.,* [6,7] demonstrate promising outcomes, however, additional analysis should be accomplished in order to identify the optimal parameters to achieve high accuracy and validate the outcomes of other related works. In this mind-set, we envision a ML assisted framework for mobile apps classification based on their intrinsic properties that is capable of (a) demonstrating ML based solutions capacities; currently it is not straight forward to compare different ML approaches and validate their outcomes, and (b) detecting malware with high accuracy.

In this work we set up the foundations for developing such a framework with emphasis on Android OS. More specifically, the proposed architecture relies on reverse engineering any given app for extracting through static analysis app's features such as activities, services, broadcasts, receivers, intent categories, APIs, and permissions that retrofit ML algorithms to characterize whether or not the examined app is malicious. We analyze the effectiveness of two well-known ML *i.e.,* k-NN and SVM to detect malware using different set of features over a data set of 2620 applications, equally divided among malware and goodware, and we provide a comparison with other related works. Further, we study the efficacy of applying a statistical analysis to the extracted features, which transform the initial set of features in a smaller dimensional space. We discuss the relevance of the different features using various selection approaches in order to identify the optimal parameters. Results indicate that we can reach accuracy up to 97%, by using optimal features with low overhead.

The rest of this paper is structured as follows. We briefly describe the related works in Sect. 2 and we introduce our ML based approach for malware detection in Sect. 3. We present the experimental evaluation and discuss our outcomes in Sect. 4. Finally, we conclude our work and we present possible future developments in Sect. 5.

2 Related Work

In this section we overview related works that rely on ML algorithms for detecting malware targeting the Android platform using as features data that can be extracted by static analysis of the app. We review, only the most relevant results to the approach proposed in this paper. This means that we consider only works that rely on features such as Application Programming Interfaces (APIs), system calls, permissions.

Applications Programming Interfaces and System Calls: A major group of Android malware detection techniques, are using systems calls or/and API calls. System calls have been used extensively for malware detection on personal computers [8–10]. So similar approaches have been introduced for detection malware in Android. More specifically, authors in [11] introduce a solution for detecting, among the others, polymorphic malware by monitoring system calls. Authors, assess the effectiveness of SVM algorithm over a dataset of 150 apps and and indicate accuracy up to 90%. Similarly, in [12] authors describe a malware detection tool called MALINE. In their solution they use as features system calls frequency as well as the corresponding call graphs. MALINE for the classification relied on SVM, Random Forest, LASSO and Ridge Regularization. Depending on the feature selection and the employed classifier accuracy results ranges between 85% to 97%. SafeDroid [13] provides another solution for Android malware detection that uses APIs as a feature, instead of relying on system calls, for retrofitting ML. It consists of the features extraction and the classification reporting services. SafeDroid considers in its analysis 743 APIs that are most frequent to malware apps, and demonstrates detection accuracy 99%, 98% and 97% for Random Forest, K-NN and SVM respectively. In a more coarse grained approach introduced in [14] authors make an analysis of ML classifiers efficiency for detecting malware using as a feature APIs packages. Authors have evaluated their idea using three well known algorithms *i.e.,* SVM, J48 and Random Forest, over 205 benign and 207 malware apps. Result indicate that detection accuracy can be as high as 92% for SVM and Random Forest, while for J48 reaches up to 89%.

Permissions: Permissions are an important factor for the proper operation of any given app, and reflect the access to sensitive resources as they have been defined by Android OS. For that reason, permissions have been considered as an alternative source of features for detecting malware in Android. Authors in [15] assess the accuracy of seven well known classifiers using as a feature app's permission on a data set consisted of 200 malicious and 200 benign Android apps. In this setup, authors demonstrate for SVM accuracy up to 95%, while for the remaining classifiers the accuracy levels are 88.6% for C4.5, 91,6% for JRip, 82.5 for Naive Bayes (NB). Permission risk rank solution introduced in [16] studies to what extend Android malware can be detected based on the permission they define in app's manifest. To do so authors first select the riskiest permission set (*i.e.,*, the most relevant from a statistical point of view) by employing statistical techniques such as T-test, and Principal Component Analysis (PCA) and secondly evaluate ML classifiers using the riskiest permission. Authors demonstrate accuracy as high as 99% for SVM, Decision Tree, and Random Forest under specific configuration using a high volume data set. In the same mind-set, the APK Auditor [17] relies on logistic regression, while evaluates its performance using a dataset of 6900 malware and 1850 benign apps. Results indicate accuracy up to 88%. Similarly, authors in [18] rely on different approaches (Gain Ratio Attribute Evaluator, Relief Attribute Evaluator) to select the optimal features to use for detecting malware with the support of machine learning algorithms. Authors evaluate their solution over a data set of 3784 apps, however, they do not report which of them was malware. The demonstrated results show accuracy up to 94%.

Other Features and Possible Combinations: One of the earliest works that combines APIs and permissions was introduce in a solution named DroidAPIMiner [7], in which well-known classifiers were assessed over on a dataset of 20000 and 3987 benign and malware apps. In this set-up authors demonstrate accuracy up to 99% in the case of K-NN, while other ML classifiers perform accuracy around 96%. In the same direction, authors in [19] combine also APIs and permissions assessing the performance of Random Forest, SVM, and Neural Networks. Authors evaluate their approach using a dataset of 5000 goodware and 1260 malware apps, and demonstrate accuracy up to 94% for all the classifiers depending on the setup. An extended set of features, such as intents, permissions, system commands, suspicious API calls, were used as input to ML in [20]. Authors accomplish an assessment different classifiers using an extended data set consisted of 18.677 malware and 11.187 benign apps correspondingly; their analysis indicates an F-score up to 0.96. In an earlier work Drebin [6] presents a holistic approach for malware detection based on SVM by using a the broadest set of features (Hardware components, Requested Permissions, App components, Filtered intents, Restricted API calls, Used permissions Suspicious API calls, Network addresses) in comparison with other related works. In the same mind set, as the other related works, Drebin evaluates its performance on the biggest public available dataset (consists of 123453 benign and 5560 malware apps). Drebin results shows accuracy 94% in the case of SVM.

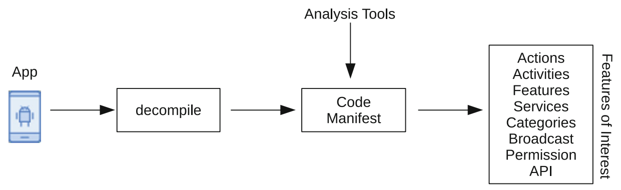

Fig. 1. High level architecture for feature extraction

3 Methodology

3.1 Overview

Any ML based approach for malware detection for Android platform consist of three phases: (a) apps collection, (b) feature extraction, and (c) apps' classification. This procedure is a standard textbook process [21] and is followed in any solution that builds on the advantages of ML.

So, similar to other ML approaches, to study the effectiveness of ML on malware detection for Android OS, firstly a collection of goodware and malware apps should be developed. To compile the appropriate datasets, we follow two different approaches. Briefly, we built a specific tool for downloading and getting access to a numerous of goodware from Google play, while for malware we made a literature research for public available data sets, as currently there is no central repository available that can be used to download them.

Secondly, towards a large scale analysis framework for ML based detection we develop the appropriate tools to automate and parellelize the experiments; especially for extracting the features of interest. Figure 1 illustrates the high level approach for extracting the features.

More specifically, we decompile any given app using the apktool[2] in order to get access to app's (original) resources, that is, among the others: (a) the manifest, (b) resources (images, fonts), (c) the source code, and (d) libs. In this work we focus on (a) app's manifest and (b) source code as they contain the core information for app's execution, however, we are planning to incorporate the remaining resources in our model in a future work. Our tools are able to distinguish the resources and analyze them in order to extract the features of interest and compile the corresponding vectors required in the third phase.

Thirdly, the classification phase takes place in order to identify whether a give app is a goodware or malware using the extracted features. In this last phase, apps' classification procedure relies on the data set of features extracted in the second phase to retrofit ML both for training and testing.

3.2 Feature Set

As various features can be used in classification, similar to other related work, we rely on features that can be extracted from any Android app namely actions, fea-

[2] https://ibotpeaches.github.io/Apktool/.

tures, services, categories, APIs, and permissions. However, based on the current related work analysis there is not any indication which of the available features are the optimal choices for Android malware detection. So at the classification phase we employ also a selection feature procedure to identify the optimal ones. In our approach for each and every feature (i) we assume that there is a vector F_i of n dimensions, where i is the type of feature. Each dimension of a vector corresponds to a binary variable which is set to 1 in case that the app contains an instance of the feature i, otherwise to 0. An app A(j) gain access a subset of these features sets F_i depending on its capabilities. In a formal way, each vector F_i is calculated using the formula 1. It should be mentioned that instead of applying the ML classification directly to raw features the statistical properties variance and sum are employed in the initial feature set. Table 1 overviews the features used to retrofit ML for the classification.

$$F_i = \{X_0, X_1, X_2 \ldots X_n\} \tag{1}$$

Since it is not known a priori which features are the most relevant feature, feature selection is a necessary step for any ML assisted detecting framework. Based on the related work there are not indications which are the dominant features for detecting malware on Android. The goal is to reduce the dimension of the data set to improve the classification time but without sacrificing the performance beyond a certain threshold. Most of the feature selection algorithms can be divided in two broad categories. The first one determines features significance using a ML algorithm that is to ultimately be applied to the data, while the second one estimates features dominance by using heuristics based on general characteristics of the data. The formers are referred to as wrappers and the latter as filters. In this paper, we assess both approaches for identifying the optimal features. For the wrapper approach we rely on the ML classifier SVM [21], and for filter we use the RELIEF algorithm proposed in [22].

Table 1. Used features for ML classification

ID	Feature description
1	Variance of actions
2	Variance of features
3	Variance of services
4	Variance of categories
5	Variance of API
6	Variance of permissions
7	Sum of actions
8	Sum of features
9	Variance of services
10	Variance of categories
11	Variance of API
12	Variance of permissions

3.3 Machine Learning Classification

Various ML algorithms are available in literature that can be used in binary classification. Recall that our goal is to provide ML based platform capable of determining whether an app is malware or not with a certain degree. So, in this preliminary work two well known ML algorithms have been used for assessing the effectiveness of ML in this domain; the K-Nearest Neighbour (K-NN) and the Support Vector Machine (SVM). We do not go into details about ML classifiers, and we refer the interested reader to relevant state-of-the-art sources such as [21].

K-NN: is probably among the most simple ML algorithms that classifies new instances based on a similarity function *e.g.,* Euclidean, Manhattan, in comparison with instances used in the training phase of classification. Considering a dataset of instances *i.e.,* the features of the goodware and malware apps in our case, each instance is classified to belong to a specific class by using the majority of votes from its neighbours, with the case being assigned to the class most common amongst its K nearest neighbour measured by the distance function.

SVM: is capable of defining a model, based on labeled data, for classifying a new instance to a specific class. To do so, SVM finds a linear separating hyperplane, that is the decision boundary, with an optimal margin among the different classes. A popular kernel, which used in this paper also, is the Radial Basis Function (RBF) kernel.

4 Evaluation

To test the accuracy and efficiency of the classifiers presented in the previous section, we performed a comparative analysis of different configurations and applied them on the collected datasets. There are two aspects to validate with regard to our work, namely the validity of (a) relevant features, and (b) the classification model and process itself.

4.1 Data Set Configuration

In our evaluation two data sets were used. The first data set (named Dataset-1) was balanced and composed by two subsets of 1300 goodware and malware apps respectively. We used this dataset to identify the optimal parameters *i.e.,* the most relevant features and evaluate ML accuracy. We assess also the effectiveness of K-NN and SVM classifiers using an unbalanced dataset (named Dataset-2) which consist of 31000 goodware and 1300 malware apps, and relying on optimal parameters that have been selected from the first dataset.

For classifiers evaluation we used a 10-fold approach where each collection of statistical features is divided into ten blocks. Nine blocks from each set of data are used for training and one block is held out for testing. The training and testing process is repeated ten times until each of the ten blocks has been held out and classified. Thus, each block of statistical features is used once for classification and nine times for training.

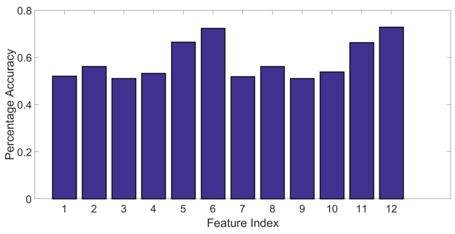

Fig. 2. Features relevance based on a wrapper approach using the SVM

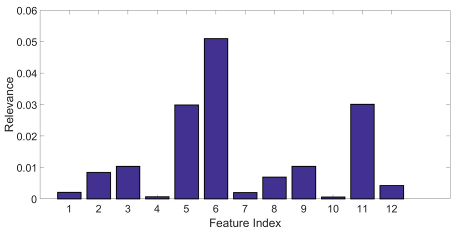

Fig. 3. Predictor relevance weight on the basis of the RELIEF algorithm

4.2 Classification Metrics

To evaluate the performance of a classifier one might use different types of metrics. For the needs of this paper we rely on accuracy that is defined as the percentage of correctly classified instances over the total instances, and is calculated according to the formula 2.

$$Accuracy = TP + TN/TP + TN + FP + FN \tag{2}$$

4.3 Results

At the first step of our approach we should select the dominant features following both a wrapper and a filter approaches as briefly described in Sect. 3). In the first case we use the SVM, while for the second we rely on RELIEFF algorithm. Figures 2 and 3 reports on the features with the highest relevance for both approaches respectively.

More specifically, we employ SVM over every single feature to identify their importance in malware detection. The values of the optimization parameters in this specific case are BoxConstraint = 5 and Scaling Factor = 5. According to this approach the dominant features are the variance and the sum of APIs and permissions in which accuracy ranges between 70% to 75%. To validate this outcome we employ also a filter selection approach by using the RELIEFF algorithm. In that case outcomes demonstrate that the variance of APIs and permissions, as well as the sum of APIs are the features with the highest relevance. In this point it should be noted that none of the related works identify the most relevant features that should be used as input to ML. The only work that select the dominant features is proposed in [16], nevertheless, they focus only on permissions.

So considering the outcomes of the two feature selection approaches we opt the variance and the sum of APIs and permissions as the features to be used for retrofitting ML classifiers, that are the $[5, 6, 11, 12]$ of the Table 1. Using this set of features we evaluate the performance of both the SVM and K-NN. Our preliminary results demonstrate (see Fig. 4) accuracy up to 95% for SVM and

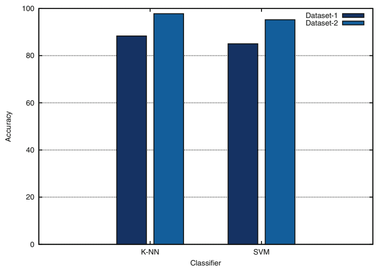

Fig. 4. K-NN and SVM accuracy for the dominant features [5, 6, 11, 12].

up to 97% for K-NN in the case of the extended dataset (Dataset-2). This means that the dataset size influence the detection accuracy. For both ML algorithms K-NN and SVM the performance is enhanced approximately to 10%. However, additional tests should be taken place for validating this outcome. For instance how the size of malware subset influence the accuracy.

Although some of the related works (*i.e.,* Safedroid [13,16], *etc.*) report accuracy up to 99%, we believe that our approach, relying on statistical features, can be an alternative option for detecting malware on Android. This is because the related works use the initial data as input to ML classifiers instead of depending on statistical transformation, and thus they perform on higher dimensions that on one side offer higher accuracy. On the other side, such approaches introduce additional processing overhead for the classification. So as there is always a trade-off between security and performance our initial outcomes demonstrate a promising solution without sacrificing accuracy. We are planning to provide a detailed analysis with regard to processing overhead that ML classifiers face in a future work.

5 Conclusions and Future Work

Apps intrinsic properties can be a valuable ally, among the others, in order to identify apps that tend to have malicious behaviour. In this paper we have described the main components for evaluating ML classifiers accuracy for detecting malware on Android platform. In our approach, we rely on the statistical properties *i.e.,* variance, mean of the initial selected feature set, and we opt the most dominant features by using the outcomes of Relief algorithm and SVM classifier. This way, ML classifiers operates on a lower dimension space, and thus less overhead can be introduced. Our initial results show accuracy up to 97%.

Though our results are promising other related works demonstrate accuracy up to 99%. For that reason, we foresee additional tests and configurations in order to reach this level of accuracy without imposing high overhead. Furthermore, the current analysis should be supported with other features that might be critical for malware detection such as (a) Cryptographic libraries, (b) application source code update, and (c) dynamic code loading presence.

References

1. Statista: Mobile Internet Usage Worldwide. https://www.statista.com/topics/779/mobile-internet/
2. Abadi, M., Budiu, M., Erlingsson, Ú., Ligatti, J.: Control-flow Integrity. In: Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, pp. 340–353. ACM, New York (2005)
3. Portokalidis, G., Keromytis, A.D.: Fast and practical instruction-set randomization for commodity systems. In: Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp. 41–48. ACM, New York (2010)
4. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. **32**(2), 5:1–5:29 (2014)
5. Ma, K., Liu, M., Guo, S., Ban, T.: MonkeyDroid: detecting unreasonable privacy leakages of android applications. In: Arik, S., Huang, T., Lai, W.K., Liu, Q. (eds.) ICONIP 2015. LNCS, vol. 9491, pp. 384–391. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26555-1_43
6. Arp, D., Spreitzenbarth, M., Gascon, H., Rieck, K.: DREBIN: effective and explainable detection of android malware in your pocket (2014)
7. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in android. In: Zia, T., Zomaya, A., Varadharajan, V., Mao, M. (eds.) SecureComm 2013. LNICST, vol. 127, pp. 86–103. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-04283-1_6
8. Madani, P., Vlajic, N.: Towards sequencing malicious system calls. In: 2016 IEEE Conference on Communications and Network Security (CNS), pp. 376–377, October 2016
9. Patanaik, C.K., Barbhuiya, F.A., Nandi, S.: Obfuscated malware detection using API call dependency. In: Proceedings of the First International Conference on Security of Internet of Things, SecurIT 2012, pp. 185–193. ACM, New York (2012)
10. Alazab, M., Venkatraman, S., Watters, P., Alazab, M.: Zero-day malware detection based on supervised learning algorithms of API call signatures. In: Proceedings of the Ninth Australasian Data Mining Conference - AusDM 2011, vol. 121, pp. 171–182. Australian Computer Society, Inc., Darlinghurst (2011)
11. Wahanggara, V., Prayudi, Y.: Malware detection through call system on android smartphone using vector machine method. In: 2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec), pp. 62–67, October 2015
12. Dimjašević, M., Atzeni, S., Ugrina, I., Rakamaric, Z.: Evaluation of android malware detection based on system calls. In: Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, IWSPA 2016, pp. 1–8. ACM, New York (2016)
13. Goyal, R., Spognardi, A., Dragoni, N., Argyriou, M.: SafeDroid: a distributed malware detection service for android, pp. 59–66, November 2016
14. Westyarian, Rosmansyah, Y., Dabarsyah, B.: Malware detection on android smartphones using API class and machine learning. In: 2015 International Conference on Electrical Engineering and Informatics (ICEEI), pp. 294–297, August 2015
15. Milosevic, N., Dehghantanha, A., Choo, K.K.R.: Machine learning aided android malware classification. Comput. Electr. Eng. **61**(Suppl. C), 266–274 (2017)

16. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
17. Talha, K.A., Alper, D.I., Aydin, C.: APK auditor: permission-based android malware detection system. Digit. Investig. **13**(Suppl. C), 1–14 (2015)
18. Pehlivan, U., Baltaci, N., Acartürk, C., Baykal, N.: The analysis of feature selection methods and classification algorithms in permission based Android malware detection. In: 2014 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 1–8, December 2014
19. Qiao, M., Sung, A.H., Liu, Q.: Merging permission and API features for android malware detection. In: 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), pp. 566–571, July 2016
20. Fereidooni, H., Conti, M., Yao, D., Sperduti, A.: ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5, November 2016
21. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Series in Data Management Systems, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2005)
22. Robnik-Šikonja, M., Kononenko, I.: Theoretical and empirical analysis of ReliefF and RReliefF. Mach. Learn. **53**(1), 23–69 (2003)