Comparative Analysis of Pre- and Post-Classification Ensemble Methods for Android Malware Detection

Shikha Badhani[1(✉)] and Sunil K. Muttoo[2]

[1] Maitreyi College, University of Delhi, Delhi, India
sbadhani@maitreyi.du.ac.in
[2] Department of Computer Science, University of Delhi, Delhi, India
skmuttoo@cs.du.ac.in

Abstract. The influence of portable devices in our day-to-day activities is of a concern due to possibilities of a security breach. A large number of malwares are concealed inside Android apps which requires high-performance Android malware detection systems. To increase the performance, we have applied ensemble learning at feature selection level (pre-classification) and at prediction level (post-classification). The features extracted are the API classes and for generating the model, extreme learning machine (ELM) has been used. The filter feature selection methods employed are Chi-Square, OneR, and Relief. The experimental results on a corpus of 14762 Android apps show that ensemble learning is promising and results in high performance as compared to the individual classifier. We also present a comparison of the pre- and post-classification ensemble approaches for the Android malware detection problem.

Keywords: Android malware · Extreme learning machine · Static analysis
Feature selection · Ensemble

1 Introduction

Android has become the fastest-growing mobile OS. According to Web analytics firm StatCounter [1], in March 2017, Android dominated the worldwide OS internet usage market share with 37.93% shooting ahead of even Windows. The esteem of Android OS also reminds us of the influence of portable devices in our lives. These devices have become the storehouse of the details that we provide to the various apps that manage our day-to-day activities such as scheduling events, online shopping, chatting, etc. Virginia Tech researchers [2] have recently discovered that various apps which we use in our mobiles have been collaborating to trade information secretly. This may result in a security breach. Even the latest initiative for enhancing Android security, Google Play Protect [3], a new built-in antivirus program, failed in the Android antivirus tests conducted by independent German lab AV-TEST [4]. It detected only 65.8% of the latest malware and just 79.2% of month-old malware. Such weak detection rates themselves highlight the need for more robust Android malware detection tools. According to internal AV-TEST statistics [4], over 18 million malware

M. Singh et al. (Eds.): ICACDS 2018, CCIS 906, pp. 442–453, 2018.
https://doi.org/10.1007/978-981-13-1813-9_44

samples are concealing in Android apps. Also, many third-party application stores provide more space for spreading malware.

The above-mentioned reasons were enough to motivate us to experiment and explore new methods for detecting Android malware. Most of the Android malware detection systems depend on three methods - static analysis, dynamic analysis, and their hybrid variants. Static analysis has an edge of over dynamic analysis because it is performed in a non-runtime environment and its ability to detect issues early before the app is executed. However, dynamic analysis may reveal the concerns that could not be detected during static analysis.

Various machine learning algorithms have been applied to detect Android malware using static features [5–7]. ELM has been explored in the past for Android malware detection [8–10]. However, in our study, we explore the effect of filter feature selection methods and their ensembles on the performance of ELM. Also, the features used in our study are API classes which differ from the ones used earlier (permissions, API calls, binder, memory, battery, CPU, network, Dalvik instructions) along with ELM.

The API classes as per Android API level 26 [11] are in thousands which constitute the features used in our study. It has been shown that insignificant features may be removed without affecting the performance of the neural network based Intrusion Detection Systems [12]. Thus, feature selection is an important step which not only eliminates useless features but also results in faster execution and simplification of the machine learning model [13]. Feature selection methods are categorized as filter methods, wrapper methods and embedded methods. We use the filter methods for selecting relevant features as they execute directly on the dataset and are not affected by the biases of any classifier. Also, they are fast as compared to other methods.

As compared to single machine learning model, combining the output of multiple prediction models known as ensemble learning, has been observed to achieve better performance [14]. Apart from classification, ensemble approach has also been applied to feature selection. In [15], five different filters were used to select a different subset of features which were used to train and test different classifiers and their outputs were combined using simple voting. In another study [16], three feature selection methods were combined based on union, intersection and multi-intersection techniques. In our work, we have compared the effect of performing ensemble at feature selection level and at prediction level. Our aim is to achieve better accuracy. We experiment with various filter feature selection methods and their ensembles and present a comparative study of their effect on the accuracy of ELM on a corpus of 14762 Android apps.

The rest of the paper is organized as follows. Section 2 describes filter feature selection methods. Section 3 introduces ELM. In Sect. 4, we present the research design. Section 5 presents the experimental results and analysis. Finally, we conclude in Sect. 6 along with possible future work.

2 Filter Feature Selection Methods

The essence of filter feature selection methods is that they rely on a statistical measure and use a feature ranking technique as the core criteria for feature selection by ordering. Filter methods are applied prior to the classification process to filter out less important

features. The relevance of a feature is measured in terms of its usefulness in differentiating between classes. Then, a ranking is generated based on the relevance and subsets of features are selected according to a threshold value. Of the wide range of filter feature selection methods available, we use the following three methods which can be applied to categorical/binary data as the features extracted in our study are binary.

- Chi-Square [17] measures the extent of independence between a feature and a class and can be compared to the chi-square distribution with one degree of freedom to judge extremeness. The initial assumption is that the feature and class are independent of each other. A high score of chi-square signifies the dependence between feature and the class and thus, the relevance of the feature.
- OneR [18] builds one rule for each feature in the dataset i.e., it learns from a one-level decision tree and then ranks features based on the fact that features which result in more accurate trees are considered to be more relevant.
- Relief [19] algorithm works by repeated random sampling of an instance from the dataset, computing its nearest neighbor from the same class as well as from different class and then calculating the worth of a feature based on its ability to discriminate between instances from different classes.

3 Extreme Learning Machine

In this section, we describe the ELM concept. Feedforward neural networks have been used extensively in the past decade due to their capability of approximating complex nonlinear mappings directly from input samples and providing models for various artificial and natural events [20]. However, one of the major bottlenecks in using feedforward neural networks is their learning speed which is slower due to the slow gradient-based learning algorithms that are used to train neural networks and the iterative tuning required for all the parameters of the networks [21]. ELM [21, 22] was originally proposed for training single hidden layer feedforward neural networks (SLFNs). The core concept of ELM is: the hidden layers of SLFNs need not be tuned but can be randomly assigned independently and a simple generalized inverse operation of the hidden layer output matrix can be used to determine the output weights of the network [21]. Since there is no iterative tuning involved like in gradient descent based learning algorithms, ELM is fast and easy to implement. ELM also intends to reach smallest training error and smallest norm of output weights [21, 22]. Apart from the above advantages, studies have shown that it has less computational complexity, nominal optimization constraints, better scalability, and generalization performance.

Given N unique samples of input data, the SLFN with L hidden nodes (additive or RBF nodes) is represented as:

$$f_L(x) = \sum_{i=1}^{L} \beta_i G(c_i, a_i, x) \tag{1}$$

where (c_i, a_i) are the learning parameters of the i^{th} hidden node, β_i is the weight vector linking i^{th} hidden node to the output node, $G(c_i, a_i, x)$ is the output of the i^{th} hidden node w.r.t the input x.

The fact that standard SLFNs with L hidden nodes can approximate the N input samples (x_j, t_j) ϵ R^n x R^m with zero error implies that there exist β_i, c_i, and a_i such that

$$\sum_{i=1}^{L} \beta_i G(c_i, a_i, x) = t_j, \quad j = 1, 2, \ldots, N \tag{2}$$

The above equation can be compactly written as

$$H\beta = T \tag{3}$$

where,

$$H(c_1, \ldots c_L, a_1, \ldots a_L, x_1, \ldots x_N) = \begin{bmatrix} G(c_1, a_1, x_1) & \cdots & G(c_L, a_L, x_1) \\ \vdots & \ddots & \vdots \\ G(c_1, a_1, x_N) & \cdots & G(c_L, a_L, x_N) \end{bmatrix}_{N \times L}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m}, \quad \mathrm{T} = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m}$$

β^T is the transpose of vector β. H is called as the hidden layer output matrix [23]. The i^{th} row of H is the output vector of the hidden layer w.r.t input x_i and the j^{th} column of H is the j^{th} hidden node's output vector w.r.t inputs x_1, x_2,…, x_N. Now the hidden node parameters c_i and a_i need not be tuned and may be assigned randomly. It has been proved in theory [22, 24, 25] that SLFNs with randomly chosen additive or RBF hidden nodes have the potential of universal approximation. Thus, independent of the training data, the hidden nodes can be generated randomly, i.e., for N unique samples of training data, randomly generated L ($\leq$ N) hidden nodes, the output vector T and output matrix of the hidden layer H comprise a linear system and the output weights β are estimated as:

$$\hat{\beta} = H^+ T \tag{4}$$

where H^+ is the Moore-Penrose generalized inverse [26] of H. Thus, the output weights can be calculated in a single step without any iterative tuning of any control parameters.

The ELM algorithm can be summarized as follows [21]:

Given a training set $\{(x_i, t_i)\}_{i=1}^{N} \subset R^n \times R^m$, the hidden node output function $G(c_i, a_i, x)$ and L hidden nodes:

1. Randomly assign hidden node parameters (c_i, a_i), i = 1,2,……, L;
2. Calculate output weight vector β: $\beta = H^+T$.

4 Research Design

The research framework of this study is shown in Fig. 1. Our work uses static features (API classes). Three filter feature selection methods (Chi-Square, OneR, and Relief) are used to generate three subsets of features which were combined by taking the union of them for the simple reason of using the maximum possible relevant features and then classification is performed on this combined subset. This is referred to as pre-classification ensemble approach and the rationale behind this approach is to take advantage of the strengths of individual selectors and release the burden of deciding which feature selection would work for a domain [27]. In another experiment, the feature subsets generated by the three filter feature selection methods are used to train and test three classifiers whose outputs are aggregated using ensemble technique of majority vote which is referred to as the post-classification ensemble approach and the concept behind this approach is that combining multiple classifiers results in more robust solutions [28] and if the features selected by each feature selection methods are diverse, so will be their classifications and hence, the better the ensemble would be. In the majority voting, every classifier makes a prediction (votes) for each instance of the test set and the final prediction is the one that receives maximum votes. The classifier used in all experiments must be unique in order to make comparisons and we use the ELM classifier for the same. Ensemble learning has been used earlier for Android malware detection [6, 8, 29, 30] to improve the performance of the model since it is based on the concept of diversity and generalization. In our study, we compare the introduction of diversity at the feature selection level vs prediction level.

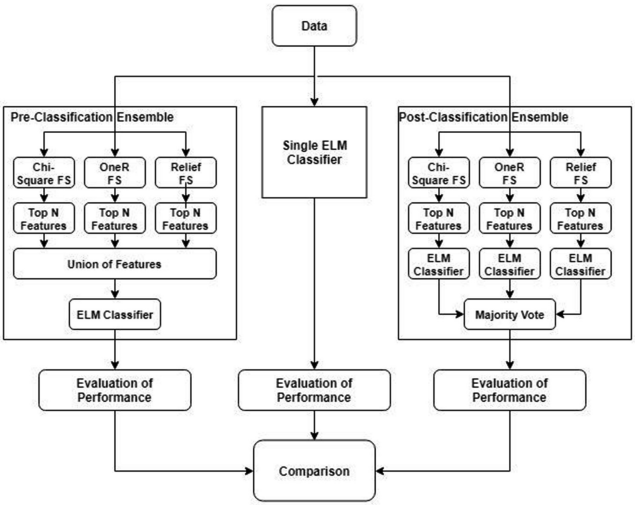

Fig. 1. Research framework

4.1 Dataset Preparation

Dataset is prepared using 7381 benign and 7381 malicious Android apps. The benign dataset is downloaded from Google play store [31] and the malicious dataset is constituted from samples collected from various sources (AndroTracker [32], Drebin database [33], Virus Total [34]). The dataset is partitioned into the training set (70%) and the test set (30%). To generalize the performance, 10-fold cross-validation was performed while generating classifier models in our experiment.

4.2 Feature Extraction

The Android API level 26 consists of 4140 API classes. Thus, we set the corresponding API class feature to 1 if an API belonging to that class is used in an app. For reverse engineering the Android app, Androguard tool [35] is used. Thus, a total of 4140 binary features are extracted.

4.3 Implementation Details

Firstly, features having zero variance are removed from the dataset as they contribute nothing to classification. After removal, we were left with 2392 features. Now, three filter feature selection methods (Chi-Square, OneR, and Relief) are applied on these 2392 features to generate three feature rankings. Most works in the previous research use several thresholds that hold different percentages of most relevant features [36]. However, the thresholds are dependent on the dataset being used. For our study, we use six different threshold values to reduce the dimension of data, including $\log_2(n)$ threshold, where n is the total number of features, following recommendations from literature to select $\log_2(n)$ metrics for software quality prediction [37]. The other five are the top 1%, 5%, 25%, 50%, and 75% of the most relevant features of the final ordered ranking obtained from each filter feature selection method. The classifier used in all the experiments is ELM.

In the pre-classification ensemble approach, we take union of the feature subsets for each of the six thresholds, generated by the three filter feature selection methods, which is then used for classification.

In the post-classification ensemble approach, each feature subset generated is used separately for classification thus resulting in three individual classifiers. Then, the output of each classifier is combined by using the majority vote.

R statistical software [38] is used to perform the experiments. For feature selection, we used the FSelector package [39] and for performing classification using ELM, we used the elmNN package [40].

4.4 Evaluation Measures

In this research, to assess the effectiveness of our proposed system, we analyzed the following measures: Accuracy (percentage of correctly identified applications); precision (percentage of actual malicious apps amongst the predicted malicious apps); recall (percentage of actual malicious apps predicted amongst the total malicious apps); F-

measure (harmonic mean of precision and recall); and area under Receiver Operating Characteristic (ROC) curve. ROC curve is a graphical plot that illustrates how a diagnostic ability of a classifier change as the internal threshold changes. The area under ROC curve summarizes the performance of a classifier in a single number. It varies between 0 and 1. As it reaches 1, the classifier has better performance.

5 Experimental Results and Analysis

In our experiments, we apply the proposed framework to predict Android malware. In this section, we present the performance measure scores achieved by 10-fold cross-validation criterion of the following experiments - single ELM classifier, three individual feature selection based ELM classifiers (Chi-Square ranked features+ELM, OneR ranked features+ELM, Relief ranked features+ELM), pre-classification ensemble and lastly, post-classification ensemble. The accuracy, precision, recall, F-measure, and area under ROC curve for six different thresholds values of selected features are shown in Tables 1, 2, 3, 4 and 5 respectively.

Table 1. Comparison of prediction accuracy

Feature threshold	Chi-Square +ELM	OneR +ELM	Relief +ELM	Pre-classification ensemble	Post-classification ensemble
Log_2 (n = 2392)	0.9332	0.9338	0.8598	0.9438	0.9338
1%	0.9377	0.9363	0.9097	0.9496	0.9379
5%	0.9399	0.9298	0.9404	0.9429	0.9417
25%	0.9352	0.9246	0.937	0.9402	0.9388
50%	0.9379	0.9189	0.9404	0.9313	0.9415
75%	0.9336	0.9329	0.9343	0.9205	0.9404
100%	0.9264				

As shown in Table 1, for all the thresholds, the accuracy of the post-classification ensemble is on a higher side than the accuracies achieved by each individual feature selection method, but it is less than the accuracy achieved by the pre-classification ensemble for low threshold values. Also, the pre-classification ensemble outperforms the individual feature selection methods only for the low value of thresholds, and as the number of features increases, the accuracies show no improvement. The highest accuracy (0.9496) is achieved by the pre-classification ensemble for the top 1% of features. The prediction accuracy of single ELM classifier using all the features is 0.9264. Thus, the introduction of ensemble increased the accuracy by 2.32%. Due to the fact that there may be a lot of irrelevant features introduced in the union combination when the threshold increases, the accuracy of the pre-classification ensemble degrades. However, as we can see that the post-classification ensemble is more immune to the introduction of irrelevant features, irrespective of the threshold, it results in the increase in accuracy as compared to the individual feature selection methods.

Table 2. Comparison of precision

Feature threshold	Chi-Square +ELM	OneR +ELM	Relief +ELM	Pre-classification ensemble	Post-classification ensemble
Log_2 (n = 2392)	0.9255	0.9252	0.8752	0.9349	0.9256
1%	0.928	0.9274	0.8978	0.9426	0.928
5%	0.9395	0.9355	0.9333	0.9371	0.9386
25%	0.9303	0.92	0.9279	0.9269	0.9282
50%	0.9195	0.9106	0.9206	0.9081	0.9252
75%	0.9163	0.9158	0.9111	0.9009	0.9258
100%	0.9129				

Table 3. Comparison of recall

Feature threshold	Chi-Square +ELM	OneR +ELM	Relief +ELM	Pre-classification ensemble	Post-classification ensemble
Log_2 (n = 2392)	0.9422	0.944	0.8392	0.9539	0.9435
1%	0.949	0.9467	0.9246	0.9575	0.9494
5%	0.9404	0.9232	0.9485	0.9494	0.9453
25%	0.9408	0.93	0.9476	0.9557	0.9684
50%	0.9598	0.9291	0.9639	0.9598	0.9607
75%	0.9544	0.9535	0.9625	0.9449	0.9575
100%	0.9426				

Table 4. Comparison of F-measure

Feature threshold	Chi-Square +ELM	OneR +ELM	Relief +ELM	Pre-classification ensemble	Post-classification ensemble
Log_2 (n = 2392)	0.9338	0.9345	0.8568	0.9443	0.9345
1%	0.9384	0.937	0.911	0.95	0.9386
5%	0.94	0.9293	0.9409	0.9432	0.9419
25%	0.9355	0.925	0.9377	0.9411	0.9395
50%	0.9392	0.9197	0.9417	0.9332	0.9426
75%	0.935	0.9343	0.9361	0.9224	0.9414
100%	0.9276				

As shown in Tables 2 and 3, highest precision (0.9426) is again achieved by the pre-classification ensemble with top 1% of features but the post-classification ensemble exhibits the highest recall (0.9684) for the top 25% of features. Although the precision and recall values fluctuate for the pre- and post-classification ensembles, but the F-measure follows the same pattern as the accuracy as shown in Table 4, with the post-

Table 5. Comparison of area under ROC curve

Feature threshold	Chi-Square +ELM	OneR +ELM	Relief +ELM	Pre-classification ensemble	Post-classification ensemble
Log_2 (n = 2392)	0.9602	0.9572	0.9321	0.9792	0.9697
1%	0.9633	0.9664	0.9618	0.9864	0.9746
5%	0.9695	0.9663	0.9777	0.9805	0.9802
25%	0.9669	0.9649	0.9798	0.9788	0.9830
50%	0.9733	0.9602	0.9768	0.9723	0.9795
75%	0.9673	0.9679	0.9737	0.9665	0.9784
100%	0.974				

classification ensemble resulting in higher F-measure as compared to the individual feature selection methods for all thresholds and the pre-classification ensemble resulting in higher F-measure values for lower thresholds (highest F-measure of 0.95 for the top 1% of features) and then degrading as the threshold increases. As compared to the single ELM classifier results, the introduction of ensemble increases the precision by 2.97%, recall by 2.58% and F-measure by 2.24%. Table 5 illustrates that even area under the ROC curve increased by 1.24% for the ensemble in comparison to the single ELM classifier. The area under the ROC curve also complies with the accuracy and F-measure results with the highest area value of 0.9864 for the pre-classifier ensemble with top 1% of the features.

6 Conclusion

Android malware detection systems require classifiers with high accuracies. One of the ways of improving the accuracy has been the use of ensemble learning. In this paper, we employ ensemble learning at the feature selection level and at the prediction level to address the problem of Android malware detection. API classes are extracted from Android apps that constitute our binary feature set. At the feature selection level, we used three filter feature selection methods and combined their feature subsets by using the union combination where all the features selected by each of the three feature selection methods are used. Thus, ensemble learning is performed prior to classification and hence the name pre-classification ensemble. At the prediction level, ensemble learning is performed by taking the majority vote of the predictions of three individual classifiers trained on each of the feature subsets generated by the three filter feature selection methods. This approach is referred to as post-classification ensemble since ensemble learning is performed after classification. The significance of ensemble not only lies in the improved performance of the models as confirmed by the experimental results but it also relieves from the burden of selecting an appropriate feature selection method or classification algorithm for a specific dataset. The results indicate that the pre-classification ensemble performs good for low values of the threshold of feature subsets and then starts degrading as the threshold increases, the reason being

introduction of irrelevant features. However, the post-classification ensemble is not affected by the thresholds as the predictions are now based on majority vote of three individual feature selection methods based classifiers.

As future work, we propose to compare the pre- and post-classification ensemble strategies for other machine learning algorithms. Also, we used only those filter methods that can be applied to binary data. The same can be explored with numeric features and other filter feature selection methods.

References

1. Simpson, R.: Android overtakes Windows for first time. http://gs.statcounter.com/press/android-overtakes-windows-for-first-time
2. Loeffler, A.: Virginia Tech researchers: Android apps can conspire to mine information from your smartphone. https://vtnews.vt.edu/articles/2017/03/eng-compsci-androidapps.html
3. Google Play Protect. https://www.android.com/play-protect
4. AV-TEST: Android Security Apps Provide Better Protection than Google Play Protect. https://www.av-test.org/en/news/news-single-view/android-security-apps-provide-better-protection-than-google-play-protect/
5. Yerima, S.Y., Sezer, S., McWilliams, G., Muttik, I.: A new android malware detection approach using Bayesian classification. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications, pp. 121–128 (2013)
6. Idrees, F., Rajarajan, M., Conti, M., Chen, T.M., Rahulamathavan, Y.: PIndroid: a novel Android malware detection system using ensemble learning methods. Comput. Secur. **68**, 36–46 (2017)
7. Zhu, H.J., You, Z.H., Zhu, Z.X., Shi, W.L., Chen, X., Cheng, L.: DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing **272**, 638–646 (2018)
8. Zhang, W., Ren, H., Jiang, Q., Zhang, K.: Exploring feature extraction and ELM in malware detection for android devices. In: Hu, X., Xia, Y., Zhang, Y., Zhao, D. (eds.) ISNN 2015. LNCS, vol. 9377, pp. 489–498. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25393-0_54
9. Demertzis, K., Iliadis, L.: Bio-inspired hybrid intelligent method for detecting android malware. Adv. Intell. Syst. Comput. **416**, 289–304 (2016)
10. Sun, Y., Xie, Y., Qiu, Z., Pan, Y., Weng, J., Guo, S.: Detecting android malware based on extreme learning machine. In: 2017 IEEE 15th International Conference on Dependable, Autonomic and Secure Computing, 15th International Conference on Pervasive Intelligence and Computing, 3rd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 47–53 (2017)
11. Class Index. https://developer.android.com/reference/classes.html
12. Sung, A., Mukkamala, S.: Identifying important features for intrusion detection using support vector machines and neural networks. In: Proceedings of the 2003 Symposium on Applications and the Internet, pp. 3–10 (2003)
13. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
14. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Mach. Learn. **51**, 181–207 (2003)

15. Bolón-Canedo, V., Sánchez-Maroño, N., Alonso-Betanzos, A.: An ensemble of filters and classifiers for microarray data classification. Pattern Recogn. **45**, 531–539 (2012)
16. Tsai, C.F., Hsiao, Y.C.: Combining multiple feature selection methods for stock prediction: union, intersection, and multi-intersection approaches. Decis. Support Syst. **50**, 258–269 (2010)
17. Imam, I.F., Michalski, R.S., Kerschberg, L.: Discovering attribute dependence in databases by integrating symbolic learning and statistical analysis techniques. In: Proceedings of the 1st International Workshop on Knowledge Discovery in Databases, Washington, DC, pp. 1–13 (1993)
18. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Mach. Learn. **11**, 63–90 (1993)
19. Kira, K., Rendell, L.A.: The feature selection problem: traditional methods and a new algorithm. In: Proceedings of AAAI 1992, pp. 129–134 (1992)
20. Ding, S.F., Xu, X.Z., Nie, R.: Extreme learning machine and its applications. Neural Comput. Appl. **25**, 549–556 (2014)
21. Huang, G.-B., Zhu, Q.-Y., Siew, C.-K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 985–990 (2004)
22. Huang, G.-B.B., Zhu, Q.-Y.Y., Siew, C.-K.K.: Extreme learning machine: theory and applications. Neurocomputing **70**, 489–501 (2006)
23. Huang, G.B.: Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Trans. Neural Netw. **14**, 274–281 (2003)
24. Huang, G.B., Chen, L.: Convex incremental extreme learning machine. Neurocomputing **70**, 3056–3062 (2007)
25. Huang, G.B., Chen, L., Siew, C.K.: Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans. Neural Netw. **17**, 879–892 (2006)
26. Rao, C.R., Mitra, S.K.: Generalized Inverse of Matrices and Its Applications, vol. 7. Wiley, New York (1971)
27. Petrakova, A., Affenzeller, M., Merkurjeva, G.: Heterogeneous versus homogeneous machine learning ensembles. Inf. Technol. Manag. Sci. **18**, 135–140 (2015)
28. Dietterich, T.G.: Ensemble methods in machine learning. In: International Workshop on Multiple Classifier Systems, pp. 1–15 (2000)
29. Aswini, A.M., Vinod, P.: Android malware analysis using ensemble features. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) SPACE 2014. LNCS, vol. 8804, pp. 303–318. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12060-7_20
30. Sheen, S., Anitha, R., Natarajan, V.: Android based malware detection using a multifeature collaborative decision fusion approach. Neurocomputing **151**, 905–912 (2015)
31. Google Play. https://play.google.com
32. Kang, H., Jang, J.W., Mohaisen, A., Kim, H.K.: Detecting and classifying android malware using static analysis along with creator information. Int. J. Distrib. Sens. Netw. **2015** (2015)
33. Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., Rieck, K.: DREBIN: effective and explainable detection of android malware in your pocket. In: Symposium on Network and Distributed System Security, pp. 23–26 (2014)
34. Virus Total. https://www.virustotal.com/
35. Androguard. https://github.com/androguard/androguard
36. Bolon-Canedo, V., Sanchez-Marono, N., Alonso-Betanzos, A.: A review of feature selection methods on synthetic data. Knowl. Inf. Syst. **34**, 483–519 (2013)
37. Wang, H.: A comparative study of ensemble feature selection techniques for software defect prediction. Mach. Learn. Appl. 135–140 (2010)

38. R Development Core Team: R: a language and environment for statistical computing. The R Foundation for Statistical Computing, Vienna, Austria (2005)
39. Romanski, P., Kotthoff, L.: FSelector: Selecting Attributes. https://cran.r-project.org/package=FSelector
40. Gosso, A.: elmNN: implementation of ELM (Extreme Learning Machine) algorithm for SLFN (Single Hidden Layer Feedforward Neural Networks). https://cran.r-project.org/package=elmNN