Android Malware Detection Using Stacked Generalization

Md. Shohel Rana, Charan Gudla and Andrew H. Sung
School of Computing Sciences and Computer Engineering, The University of Southern Mississippi
Hattiesburg, MS, 39406, Unite States
(md.rana, charan.gudla, andrew.sung)@usm.edu

Abstract

Malware detection plays a key role in Android device security due to the popularity of Android with billions of active users that encouraging cybercriminals to push the malware into this operating system. The growth of malware is now becoming a serious problem. Recently, extensive research has been conducted to detect malware on Android devices using machine learning based methods profoundly depending on domain knowledge for manually extracting malicious features. In this paper, we evaluate tree-based machine learning algorithms by Stacked Generalization concept for detecting malware on Android in conjunction with implementing a substring-based method for training the algorithms. We perform experiments on 11,120 samples containing 5,560 malware samples and 5,560 benign samples provided by DREBIN dataset on 8 malware families. The evaluation results show how stacked generalization achieves 97.92% validation accuracy for malware detection on DREBIN dataset.

keywords: Machine Learning, Classifier, DREBIN, Substring, Malware, Stacked Generalization.

1 Introduction

At the present time, detection of malware is always distinctly concerned matter in Android device security field while numerous apps provide wealth resources for users but also carry certain potential threats on those devices. Android, built on the Linux Kernel is an open source mobile-based operating system and its architecture into five components and two models of permissions; (i) A sandbox environment at the kernel level and (ii) API used to expose to the user during installation of an application [1]. The assembly of an Android app consists of application code (.dex files), resources, and AndroidManifest.xml file, which provides the application's features and the security configurations information [2, 3]. After DE-compilation of an Android APK file, we study the An-droidMenifest.xml file to check whether any permission used and then numerous API functions are written to call in java file to check whether any code hiding image script available or not.

In this paper, we propose a substring-based method for feature selection for use in Android malware detection and show how to make stacked generalization work for classification tasks by addressing two crucial issues: (i) the type of attributes used to form level-1 data, and (ii) the type of level-1 generalizer in order to get improved accuracy using the stacked generalization method by evaluating tree-based machine learning algorithms to detect malware on Android on the DREBIN dataset [4]. The substring-based method helps in removing possibly irrelevant information and speeds up the detection and the stacked generalization achieves greater predictive accuracy by using high-level model to combine lower level models. The results of Decision Tree, Random Forest, Extremely Randomized Tree, and Gradient Tree Boosting are assessed based on various features including api_call, feature, url, service_receiver, permission, call, intent, real_permission, activity, provider. Analyzing the outcome of the experiments using these algorithms we found that the Stacking achieves 97.92% accuracy in Android malware detection.

The remainder of the paper is organized as follows: section 2 gives an overview of Stacked Generalization (or Stacking); section 3 describes related work; section 4 presents our proposed malware detection methodology, including dataset description, feature extraction, technology used, training and testing; section 5 presents results and analysis including performance metrics; and section 6 gives conclusions and future work.

2 Overview of Stacked Generalization

According to Wolpert [5], Stacked generalization (or stacking) is a way of combining multiple models to achieve higher predictive accuracy. The procedure of stacking is as follows:

a. Split the training set into two disjoint sets.

b. Train several base learners on the first part.

978-1-943436-13-2 / copyright ISCA, SEDE 2018
October 8-10, 2018, New Orleans, Louisiana, USA

c. Test the base learners on the second part.

d. Using the predictions from c) as the inputs, and the correct responses as the outputs, train a higher-level learner.

where steps a) to c) are the same as cross-validation, but instead of using a winner-takes-all approach, we combine the base learners, possibly non-linearly.

Given a data set $\mathrm{L} = (y_n; x_n); n = 1, \ldots, N$, where y_n is the class value and x_n represents the attribute values of the nth instance, randomly split the data into J almost equal parts $L_1, \ldots, L_J$. Define L_j and $L^{(-j)} = L - L_j$ to be the test and training sets for the jth fold of a J-fold cross-validation. Given K learning algorithms, which we call level-0 generalizers, invoke the kth algorithm on the data in the training set $L^{(-j)}$ to induce a model $M_k^{(-j)}, fork = 1, \ldots, K$. These are called level-0 models.

For each instance x in L_j, the test set for the jth cross-validation fold, let $v_k^{(-j)}(x)$ denote the prediction of the model $M_k^{(-j)}$ on x. Let,

$$z_{kn} = v_k^{(-j)}(x_n) \tag{1}$$

At the end of the entire cross-validation process, the data assembled from the outputs of the K model is

$$L_{cv} = (y_n, z_{1n}, \ldots, z_{Kn}), n = 1, 2, \ldots, N \tag{2}$$

This is the level-1 data. Use some learning algorithm that we call the level-1 generalizer to derive from this data a model M. This is the level-1 model. To complete the training process, models $M_k, k = 1, \ldots, K$, are derived using all the data in L. Now let us consider the classification process, which uses the models $M_k, k = 1, \ldots, K$, in conjunction with M. Given a new instance, models M_k produce a vector $(z_1, \ldots, z_K)$. This vector is input to the level-1 model M, whose output is the final classification result for that instance. This completes the proposed stacked generalization method [6].

3 Related Works

Talha, et al. [7] proposed a permission-based detection system 'APK Auditor' to classify the Android apps as benign or malicious and obtained 88% accuracy with a 0.925 specificity using 8762 applications containing 1853 benign applications and 6909 malicious applications.

Sahs and Khan [8] proposed a supervised machine learning technique to detect malware on Android using SVM, and where 'Androguard' tool and the Scikit-learn framework are first used to extract information from the APKs [9].

Yeima, et al. [10] proposed a model that provides indicators of potential malicious activities based on Bayesian classification using static analysis, and the best results were obtained TPR (True Positive Rate) 90.6%, FNR (False Negative Rate) 0.094%, accuracy of 93.5% and AUC (Area Under Curve) of 97.22%.

DroidDolphin [11] is a dynamic analysis framework based on machine learning to detect malware on Android. It performs analysis by extracting information from API calls and 13 activities by running the application on virtual environments and achieved a precision of 86.1% and an F-score of 0.875 by using SVM and the LIBSVM library [12].

Feizollah, et al. [13] proposed a model using dynamic analysis by applying five supervised machine learning algorithms, and the best results were obtained by KNN: TPR of 99.94% against an FPR (False Positive Rate) of 0.06%.

4 Methodology

4.1 Dataset Description

For experiments, we use the 'DREBIN' dataset that contains of 11,120 of 123,453 real Android applications, where 5,560 applications contain malware samples from 179 different malware families and 5,560 are benign samples and collected during August 2010 to October 2012 and the top 20 families of malware of this dataset is shown in Table 1.

Table 1: Top malware families of our dataset

Malware family	# Entries	Malware family	# Entries
Fake Installer	925	Adrd	91
DroidKunFu	667	DroidDream	81
Plankton	625	ExploitLinuxL otoor	70
Opfake	613	Glodream	69
Ginmaster	339	MobileTx	69
BaseBridge	330	FakeRun	61
Iconosys	152	SendPay	59
Kwin	147	Gappusin	58
FakeDoc	132	Imlog	43
Geinimi	92	SMSreg	41

4.2 Data Preprocessing and Feature Selection

a. **Data collection and balancing:** The dataset is composed of benign apps and apps infected with malware. A balanced dataset is constructed for

experiments, which is done by randomly selecting the same number of malware samples and benign samples.

b. **Substring array creation based on feature sets:** As we use the DREBIN dataset, which focuses on the manifest XML file and the disassembled (.DEX) code of the Android apps. The dataset has 8 features from two sources including:

i. Feature sets from the manifest:

- Feature 1 (Hardware components): It is a set of requested hardware components.
- Feature 2 (Requested permissions): Android permissions play vital role in security mechanism allowed by users during the installation of application. Malicious Application has trends to request dangerous permission by which it can get access to sensitive information.
- Feature 3 (App components): Activities, Services, Content providers and Broadcast receivers.
- Feature 4 (Filtered intents): Intents are performed in android inter process and intra process communication.

ii. Feature sets from disassembled code:

- Feature 5 (Restricted API calls): Restricted API calls is performed based on Android permissions allowed during installation.
- Feature 6 (Used permissions): Using this set of features, we can ensure that the requested permissions and the API function calls are directed to malicious activities or not.
- Feature 7 (Suspicious API calls): Sometimes some API functions are used by malware by which can be get access of sensitive information about device and used for obfuscation.
- Feature 8 (Network addresses): Network addresses or URLs are commonly used by malware to pass data or to execute external commands.

iii. Variation of word from feature sets: API calls, URL are unique sentences, and so we can count each as single word. But for permissions, activities, intents, services, they have multiple sequences of words, for example, android.hardware.telephony. We create 3 substrings using last 1, 2 and 3 words respectively as a meaningful information to identify the most important word in any permissions, activities, intents, services, etc.

c. **Substring array loading:** The substring array is loaded to split it 80:20 ratio to train and test the learning machine.

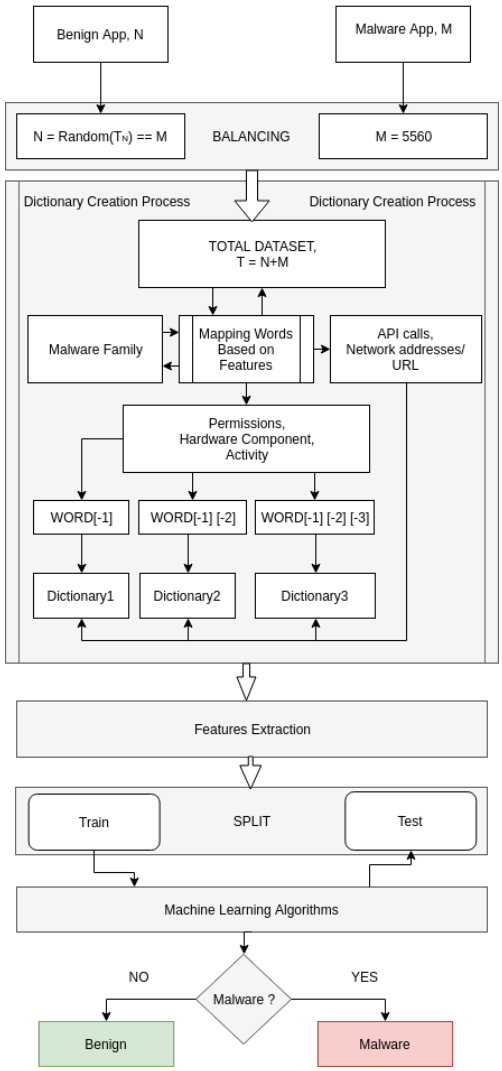


Figure 1: Substring-based malware detection method

4.3 Training and Testing

In order to obtain good results, we split our dataset into 80% for training and 20% for testing. To perform the classification task, we apply three learning algorithms used as the level-0 generalizers including Decision Tree [14], Extremely Randomized Tree [15], and Gradient Boosted Tree [16]. And we compare the effect of these three algorithms and Random Forest [17] as the level-1 generalizers. After training the model we evaluate its accuracy by testing it on new instances.

5 Experimental Results

5.1 Measurement Metrics

Common metrics are used to evaluate the performance of the various combinations of learning machines and features on the dataset, including the following: Accuracy (AC) is the proportion of the total number of corrected predictions. Precision (P) is the proportion of the correctly predicted positive cases. Recall or True Positive Rate (TPR) is the proportion of the correctly identified positive cases. False Positive Rate (FPR) is the proportion of negatives cases that were incorrectly classified as positive. f1- Score or F-Measure is a weighted average of the True Positive (TP) rate or recall and Precision (P). ROC Curve is a graph to summarize the performance of the classifier over all probable thresholds generated by plotting the True Positive (TP) Rate in Y-axis against the False Positive (FP) Rate in X-axis.

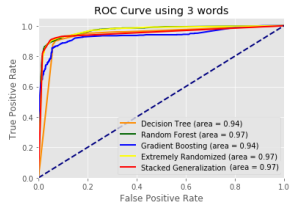
5.2 Performance Results

For the DREBIN datasets, W-fold cross-validation is performed. In each fold of this cross-validation, the training dataset is used as L, and the models derived are evaluated on the test dataset. This cross-validation is used for evaluation of the entire procedure and is quite different from the J-fold cross-validations employed as part of the stacked generalization operation. However, both W and J are set to 10 in the experiments. Table 2 shows the result of stacked generalization using the level-1 model M, for which the level-1 data comprises the classifications generated by the level-0 models M'.

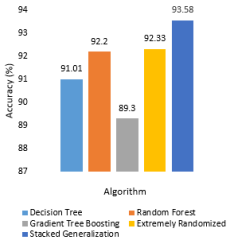
Table 2: Performance results of stacked generalization with tree-based algorithms

Substring	Algorithms	Precision		Recall		f1-score		Accuracy
		0	1	0	1	0	1	
3 words from each feature	Decision Tree	0.95	0.89	0.89	0.95	0.92	0.92	91.01
	Random Forest	0.95	0.92	0.93	0.95	0.94	0.95	92.20
	Gradient Boost	0.86	0.89	0.90	0.84	0.88	0.87	89.30
	Ext. Randomized	0.95	0.91	0.91	0.95	0.93	0.93	92.33
	Stacking	0.92	0.94	0.95	0.91	0.93	0.93	93.58
2 words from each feature	Decision Tree	0.93	0.93	0.92	0.93	0.93	0.93	91.86
	Random Forest	0.93	0.95	0.95	0.94	0.94	0.94	92.77
	Gradient Boost	0.87	0.91	0.91	0.87	0.89	0.89	89.41
	Ext. Randomized	0.94	0.94	0.94	0.94	0.94	0.94	93.18
	Stacking	0.93	0.95	0.95	0.94	0.95	0.95	94.17
1 word from each feature	Decision Tree	0.96	0.96	0.96	0.96	0.96	0.96	95.92
	Random Forest	0.97	0.98	0.98	0.97	0.97	0.97	96.06
	Gradient Boost	0.93	0.94	0.94	0.93	0.94	0.94	92.73
	Ext. Randomized	0.97	0.97	0.97	0.97	0.97	0.97	96.34
	Stacking	0.96	0.98	0.98	0.96	0.97	0.97	97.92

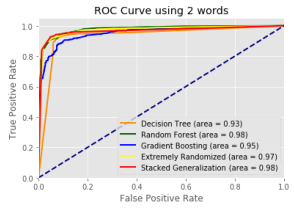
Finally, we observe that the stacked generalization as almost always producing the best results compared to the other single classifiers. We compare the overall accuracy of each tree-based machine learning algorithm using the same parameters and summarize the results in Fig. 2, where it is seen that the best performance is obtained by stacked generalization producing the overall accuracy of 97.92%.



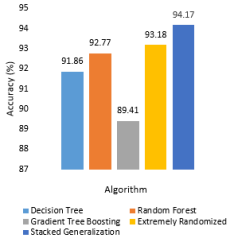
(a) ROC using 3 words



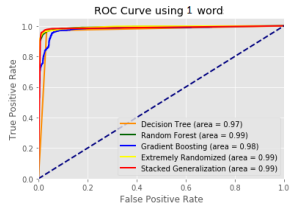
(b) Accuracy using 3 words



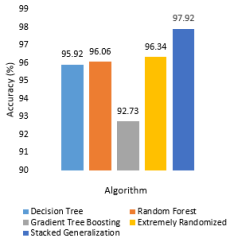
(c) ROC using 2 words



(d) Accuracy using 2 words



(e) ROC using 1 word



(f) Accuracy using 1 word

Figure 2: ROC and Accuracy curves

6 Conclusion and Future Work

Detecting mobile malware has become an important problem due to the rapid worldwide proliferation of mobile devices; accordingly, several datasets of Android malware have been created for research and analysis. Most of the previous research on Android malware used only two common features: system permission and API call. In our investigation, we used 8 features and also observed the variations of parameters and their effect on the accuracy of detection. We conducted experiments using the DREBIN dataset with successful implementation of stacked generalization in classification tasks by combining these four tree-based classification algorithms, found to achieve better predictive accuracy 97.92% based on cross-validation, thereby provides a

strong basis for building effective malware scanners.

For future work, in order to detect Android malware in real time manner, we propose to study a new protocol called DanKu [18] that utilizes blockchain technology through decentralized contracts allowing anyone to post a dataset, an evaluation function, and a reward for the best trained machine learning model. In order to model the data, contributors submit their trained networks to the blockchain by training with deep neural networks. And finally, the blockchain rewards the best submitted model by executing these neural network model.

References

[1] C. Mulliner Pau O. Fora Stephen A. Ridley Georg Wicherski Joshua J. Drake, Z. Lanier. *Android hacker's handbook*. John Wiley and Sons, 2014.

[2] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 300–305, 2013.

[3] Ping W. Yan and Zheng Yan. A survey on dynamic mobile malware detection. *Software Quality Journal*, pages 1–29, 2017.

[4] Malte Hubner Hugo Gascon Daniel Arp, Michael Spreitzenbarth and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

[5] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[6] Kai M. Ting and Ian H. Witten. Stacked generalization: when does it work? *New Zealand: University of Waikato, Department of Computer Science*, 97(03), 1997.

[7] Dogru I. Alper Kabakus A. Talha and Cetin Aydin. Apk auditor: Permission-based android malware detection system. *Digital Investigation*, 13:1–14, 2015.

[8] Justin Sahs and Latifur Khan. A machine learning approach to android malware detection. In *Proceedings of the 2012 European Intelligence and Security Informatics Conference*, pages 141–147. IEEE Computer Society, 2012.

[9] Scikit-learn: machine learning in python scikit-learn 0.16.1 documentation, last accessed: 2018-08-14. http://scikit-learn.org/stable/.

[10] Gavin McWilliams Suleiman Y. Yerima, Sakir Sezer and Igor Muttik. A new android malware detection approach using bayesian classification. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, pages 121–128. IEEE Computer Society, 2013.

[11] Wen C. Wu and Shih H. Hung. Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*. ACM Press, 2014.

[12] Chih C. Chang and Chih J. Lin. Libsvm: A library for support vector machines. *ACM TIST*, 2:1–27, 2011.

[13] Rosli Salleh Fairuz Amalina Rauf R. Maarof Ali Feizollah, Nor B. Anuar and Shahaboddin Shamshirband. A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malaysian Journal of Computer Science*, 26(4):251–265, 2013.

[14] Prashant Gupta. Towards data science: Decision trees in machine learning, last accessed: 2018-08-14. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052.

[15] Niklas Donges. Towards data science: The random forest algorithm, last accessed: 2018-08-14. https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.

[16] Ayoub RMIDI. Towards data science: Build, develop and deploy a machine learning model to predict cars price using gradient boosting, last accessed: 2018-08-14. https://towardsdatascience.com/build-develop-and-deploy-a-machine-learning-model-to-predict-cars-price-using-gradient-boosting-2d4d78fddf09.

[17] Dan Benyamin. A gentle introduction to random forests, ensembles, and performance metrics in a commercial system, last accessd: 2018-08-14. http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics.

[18] A. Besir Kurtulmus and Kenny Daniel. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain, algorithmia research, last accessed: 2018-08-14. https://algorithmia.com/static/documents/d3a4c04/Machine_Learning_Models_on_the_Ethereum_Blockchain.pdf.