Android Malware Detection and Classification using LOFO Feature Selection and Tree-based Models

S Abijah Roseline and S Geetha

School of Computer Science and Engineering, Vellore Institute of Technology - Chennai Campus, Vandalur - Kelambakkam Road, Chennai, India

Abstract. Cybersecurity threats on mobile devices are also growing substantially with the subsequent rise in the usage of smartphones and mobile applications. Cybercriminals inevitably have expanded their malicious operations to Google's Android mobile operating system. Due to the limitations of traditional signature-based approaches and the constant evolution of new malware, current malware detection systems are turned to be empowered by intelligent machine learning models. In this aspect, malware defense techniques strive to integrate data science and cybersecurity. This paper presents an Android malware detection system that incorporates the Leave One Feature Out (LOFO) approach and uses tree-based learning models to classify malware applications based on the top selected features. The experimental evaluation is conducted on the DREBIN dataset to demonstrate the efficacy of the proposed method. The prediction accuracy of the XGBoost classifier is observed to outperform other tree-based models, but with higher computational costs.

Index Terms—Android Malware Detection, Mobile Security, Machine Learning, Tree-based Models, Ensemble Learning, Feature Selection, Feature Importance.

1. Introduction

Various research methods have been reported to counter Android malware attacks. The extraction of static, dynamic, and hybrid features was used in the literature for the detection of Android malware applications. The static features are obtained using a static analysis approach that does not require an application to run. The dynamic features are extracted using a dynamic analysis approach that allows an application to run in a controlled environment and track its behavior. Static analysis is advantageous over the dynamic analysis approach as execution environments are not required and computational overheads are relatively low. Machine learning models take advantage of static or dynamic features for effective detection and classification of Android apps. A good classifier model should hold a balance between the errors of bias and variance. This is regarded as the trade-off management of bias-variance errors. This trade-off analysis is performed using Ensemble learning.

Ensemble methods are known to contribute enhancement to tree-based models. Bagging, Boosting, and Stacking are some of the widely used ensemble techniques. Bagging is an ensemble approach used to decrease the variance of the predictions by integrating the outcomes of several classifiers modeled on different sub-samples of the same dataset. Boosting combines base learners or weak learners to form a strong prediction rule. Various machine learning algorithms can be used as base learners and iteratively produces weak prediction rules. Finally, it integrates the outcomes of weak learners and produces a strong learner that ultimately increases the model's prediction ability. Several boosting algorithms provide the model's performance an extra boost. Gradient Boosting (GBM), LightGBM, AdaBoost, and eXtreme Gradient Boosting (XGBoost) are the various types of boosting algorithms.
The proposed system is capable of leveraging not only traditional singular tree learner model like Decision Trees [12] but also ensemble learning algorithms like Random Forest [13], Extra Trees Classifier [14], Gradient Boosting (GBM) [15], LightGBM [17], AdaBoost [16], and eXtreme Gradient Boosting (XGboost) [16], etc. for improved classification accuracy. The proposed system is designed to select important Android application features for effective Android malware detection by training various tree-based classifiers at the lower level. A set of tree-based algorithms are then utilized to classify the Android malware from cleanware applications at the higher level, one of which is selected to build a final model.

The rest of the paper is structured as follows. Section 2 discusses the literature survey and section 3 presents the proposed Android Malware Detection System. The experimental analysis and discussions are presented in section 4. Finally, section 5 summarizes the paper.

2. Literature Survey

Machine learning methods leverage static, dynamic, and hybrid analysis techniques to detect Android malware applications. Several significant researches in malware detection and classification are studied [6] [10] [19] [21].

Elish et al. [4] proposed a benign property enforcement approach that extracted distinct behavioral features from legitimate programs and designed the appropriate classification models. Their system checked whether the input program matched with the benign property model. The system used a complex feature known as TriggerMetric, which captured the dependence relations between user activities and sensitive operations statically. This property is referred to as user-trigger dependence. The system was evaluated using a huge number of malicious apps and free common apps. The system provided a more proactive defense than other malware detection approaches. The rule-based classification with user behavior features made the proposed approach more powerful. The system detected malware apps with high accuracy and a low false-positive rate.

Alam et al. [3] proposed DroidNative for detecting both bytecode and native code Android malware. The system used specific control flow patterns to overcome obfuscations and provided automation. The cross-platform semantic-based signatures were built at the native code level for Android. Techniques such as packing and applications that download malware cannot be properly analyzed by the system. Their system could not detect unknown, zero-day malware. The system detected known malware variants and identified zero-day malware if its control structure matched an
existing malware trace in the training database. It did not detect malware that changed the control flow of code based on a threshold value.

Shabtai et al. [1] proposed a generic and modular Host-based Malware Detection framework called Andromaly for the detection of malware in Android. Andromaly continuously monitored different features and activities occurring in the mobile device. Machine learning based anomaly detectors were employed for malware classification. Their system identified the optimal combination of a classification and feature selection method and also the observed features that contributed to the robust detection of new Android malware. Four malware applications were developed, and the efficiency of Andromaly was assessed with different feature selection methods and anomaly detection methods to detect new malware based on known malware samples. Their system was effective in malware detection on Android mobile devices. They inferred that their system failed to detect new, unknown attacks.

A sustainable learning-based malware detection is achieved by using consistent features that best discriminates malware from benign applications. Such features could be extracted from studying the growth of Android applications. H. Cai [5] studied and compared the sustainability metrics such as reusability and stability of Android malware detectors using machine learning techniques. They proposed a novel malware detector called DroidSpan that exploits the behavioral profile of Android applications by recording sensitive access distribution. They inferred that the sustainability and efficiency of their approach was better compared to the existing approaches.

Garcia et al. [7] proposed an Android malware detection and classification system called RevealDroid that does not require complex program analysis or more number of features. Their system depends on features including API calls, indicative calls characterized to the extent to which the invoked methods can be accomplished and native binary invocations to internal and external functions. The efficiency of their model were evaluated using a larger dataset with 54,000 malware and benign applications. They also assessed obfuscation resiliency of RevealDroid by applying various transformations on malware applications. Their approach is 13 times faster than other methods that use information-flow based feature extraction. Also, they outperformed other methods in terms of accuracy and obfuscation resilience. They have used balanced dataset for evaluating their model. The risk of external validity problems may occur due to less number of applications in the dataset. This leads to generalizability issue in the results. Internal validity issues may also occur due to mislabelling of samples. Construct validity is another issue due to the dataset transformations and non-realistic obfuscations.

Onwuzurike et al. [8] proposed a new Android malware detection system called MaMaDroid based on capturing behavioral models of applications by observing the API call sequences as Markov chains. The API calls were finely abstracted by their classes rather than packages or families. Their system showed better predictive accuracy for unknown samples without retraining. MaMaDroid outperformed other methods in terms of accuracy, computational time, robustness to new malware and resiliency to changes in API. In order to determine whether the performance of MaMaDroid is purely due to abstraction, a variant of MaMaDroid called Frequency Analysis Model (FAM) was developed that incorporates API call abstraction based on frequency analysis. They inferred that FAM is better than DroidAPIMiner, but not robust as MaMaDroid. Their approach has not been tested for resiliency to evasion methods like repackaged malware applications and malicious API call injection to modify Markov models.

3. Proposed Methodology
The proposed method contains two stages, namely, the feature engineering stage and the tree-based classifier evaluation stage. The overall system flow of the proposed model is shown in Figure 1. The individual stages are described in the following subsections.

3.1 System flow of the proposed method

3.1.1 Stage 1 – Feature Engineering Stage
To effectively classify malware from cleanware applications, the feature engineering stage develops an appropriate input dataset that is consistent with machine learning models and optimizes the performance of the training models. Wrapper approaches use the predictive performance of a learning model to determine the relative importance of feature subsets. The error rate estimate of the classification algorithm is a crucial step in wrapper approaches. Predictive performance is determined by computation of the classification error rate or theoretical proofs. If the classification error rate with a subset of features is lower, the subset is identified better.

For each feature, Leave One Feature Out (LOFO) importance [18] calculates the importance score for the feature set based on the tree-based learning model, Receiver Operating Characteristics-Area Under Curve (ROC-AUC) scorer and k-fold cross-validation method. First, LOFO evaluates the performance of the model for all the features in the DREBIN malware dataset. Then, each feature in the dataset is deleted iteratively and the performance of the tree model is calculated using the validation scheme and the ROC-AUC metric. The importance of a feature is assessed by analyzing the variations in the score after the elimination of the respective feature.

3.1.2 Stage 2 – Classifier Evaluation
The second stage of the proposed methodology is model selection by tree-based classifier evaluation. The classifiers are evaluated with the final set of important features obtained from the first stage. A detailed insight into various classifiers is presented in Section 4.4.

4. Test results and discussion
4.1 Test scenario
Empirical analysis with Android malware dataset was carried out to examine the efficiency of the proposed algorithm in the following perspectives:
- Classification performance metrics based on the number of top selected features
- Average running time

4.2 Dataset
The proposed approach has been assessed by experiments on the Android malware Drebin [2] dataset. It contains 15,036 samples, of which 5560 samples are malware applications and 9476 samples are cleanware applications. The samples belong to two classes (malware/cleanware) and each sample has 215 features obtained from static code analysis.

4.3 Test set-up
The tests are carried out using Intel i7, 8 GB RAM, DDR3, 1 TB hard drive on a Windows XP operating system. The proposed methodology is implemented using Python. The classification error is calculated using 10-fold cross-validation for all the experiments. The results were taken in nine settings by selecting the top 5, 15, 20, 25, 30, 35, 40, 45, 50 features, and using the selected subsets the tree-based models were assessed. The results obtained are shown in Tables 2–4.

Figure 2 shows importance values for all tree learners after creating a feature matrix. Figure 3 shows the feature importance graph for the best-performed XGBoost classifier that selected the important features to classify malware from cleanware applications. In the feature importance graph, the colored bar is the mean and the black lines denote standard deviation. The features that correspond to the green color bar indicate the important features. The features corresponding to the red color indicate the negative importance that degrades the performance of the model. From the results, “RECEIVE_BOOT_COMPLETED” is selected as the most important feature. The top selected 30 features are listed and described in Table 1.

Figure 4 shows the comparison of predictive accuracies of DT, RF, ET, GBM, AdaBoost, Light GBM, and XGBoost based on the selection of a varied number of features. With fewer features, the XGBoost model shows the highest performance among all tree-based models. Table 2 shows the performance evaluation results for Android malware Detection and Classification based on features selected using the tree-based LOFO approach. The efficiency of the proposed model is assessed using performance metrics such as Accuracy (Acc), Precision (Pr), Recall (Re) and F1-score (F1). XGBoost
shows accuracies of 93.02% for 15 top features, 94.17% for 20 selected features, 95.43% for 25 selected features, and 95.59% for 30 selected features in classifying Android malware. LightGBM is the second most effective model. By using 5 and 10 top features, LightGBM shows 81.82% and 85.37% respectively, where all other models show comparably less performance.

Figure 2. Graphs showing importances for Android Application Features using LOFO approach based on Tree learners.

Figure 3. Plot showing feature importance for best classifier XGBoost on DREBIN dataset.
Table 1. Best Android Malware features selected by XGBoost Model.

| S. No | Features                        | Type                | Description                                                                 |
|-------|---------------------------------|---------------------|----------------------------------------------------------------------------|
| 1     | RECEIVE_BOOT_COMPLETED          | Manifest.Permission | Allows the application to obtain the Intent.ACTION_BOOT_COMPLETED broadcast after the device has stopped booting. |
| 2     | WRITE_SYNC_SETTINGS             | Manifest.Permission | Allows applications to write the sync settings                               |
| 3     | android.telephony.SmsManager    | Telephony           | Manages SMS services such as sending data, text, and pdu SMS messages        |
| 4     | TelephonyManager.getDeviceId    | Telephony           | Returns the unique device identification number                             |
| 5     | ACCESS_COARSE_LOCATION          | Manifest.Permission | Enables an application to control an approximate location                   |
| 6     | chmod                            | OS                  | Change file permissions                                                     |
| 7     | System.loadLibrary              | System              | Loads the native library                                                    |
| 8     | READ_PROFILE                    | Permission          | Allows the application to read personal profile information stored on your device |
| 9     | READCALENDAR                    | Manifest.Permission | Enables an application to read calendar information from the user           |
| 10    | ACCESS_NETWORK_STATE             | Manifest.Permission | Enables applications to access networking information.                     |
| 11    | READ_PHONE_STATE                | Manifest.Permission | Enables read-only access to phone state, like current cellular network details, status of any ongoing calls, and record of all registered phone accounts on the device |
| 12    | android.os.Binder               | OS                  | Provides the standard local implementation of a remote object               |
| 13    | WRITE_APN_SETTINGS              | Manifest.Permission | Enables applications to write apn settings                                   |

Figure 4. Performance graph for tree-based models on the Android DREBIN dataset.
and read sensitive fields such as user and password from current apn settings.

| 14 | EXPAND_STATUS_BAR | Manifest.Permission | Allows an application to extend or disintegrate the status bar. |
| 15 | android.intent.action PACKAGE_REMOVED | Intent | An existing application package has been removed from the device. |
| 16 | onServiceConnected | Service Connection | Called when a connection to the Service has been established. |
| 17 | Process.start | Application Process | Allows an application to start its process. |
| 18 | WRITE_USER_DICTIONARY | Permission | Allows the application to write new words into the user dictionary. |
| 19 | WRITE_SETTINGS | Manifest.Permission | Allows the device settings to be read or written by an application. |
| 20 | bindService | Content | Connect to an application service. |
| 21 | READ_CONTACTS | Manifest.Permission | Enables an application to read the contact information of the user. |
| 22 | java.net.URLDecoder | java.net package | Utility class for HTML form decoding. |
| 23 | android.content.pm.PackageInfo | Content | Overall details on the contents of a package. |
| 24 | java.lang.Class.cast | Java.lang package | Sets an object to the class or interface represented by this Class object. |
| 25 | WRITE_CALL_LOG | Manifest.Permission | Allows the application to write (but not read) the call log information of the users. |
| 26 | java.lang.Class.getPackage | Java.lang package | Gets the package for this class. |
| 27 | android.intent.action.SET_WALLPAPER | Intent | Show settings for choosing wallpaper. |
| 28 | abortBroadcast | Content | Sets the flag indicating that this receiver should abort the current broadcast. |
| 29 | divideMessage | Telephony | Divide a text message into many fragments. |
| 30 | PackageInstaller | Content | Allows the capability to install, upgrade and uninstall apps on the device. |

Table 2. Performance Comparison of Tree-based Models for the selected number of features on Android Malware DREBIN dataset.

| Model          | Number of Features selected | 5  | 10  |
|----------------|----------------------------|----|-----|
|                | Acc (%) | Pr  | Re  | F1  | Acc (%) | Pr  | Re  | F1  |
| Decision Tree  | 81  | 0.83 | 0.6027 | 0.6983 | **85.37** | 0.869 | 0.7053 | 0.7787 |
| Random Forest  | 66.14 | 0.544 | 0.4459 | 0.4902 | 69.82 | 0.7975 | 0.2321 | 0.3595 |
| ExtraTrees     | 81.11 | 0.798 | 0.6464 | 0.7141 | 85.19 | 0.7925 | 0.805 | 0.7987 |
| GBM            | 75.99 | 0.613 | 0.9289 | 0.7385 | 76.05 | 0.6136 | 0.9289 | 0.739 |
| AdaBoost       | 81.75 | 0.9 | 0.5626 | 0.6923 | 80.22 | 0.7162 | 0.7588 | 0.7369 |
| Light GBM      | **81.82** | 0.855 | 0.6045 | 0.7082 | **85.37** | 0.7827 | 0.8293 | 0.8053 |
| XGBoost        | 77.69 | 0.701 | 0.6774 | 0.6891 | 84.59 | 0.8567 | 0.6938 | 0.7667 |

| Model          | Number of Features selected | 15 | 20 |
|----------------|----------------------------|----|----|
|                | Acc (%) | Pr  | Re  | F1  | Acc (%) | Pr  | Re  | F1  |
|                |         |     |     |     |         |     |     |     |


Table 3 shows the performance of the proposed feature selection approach and tree-based classifier models in terms of computational time. The feature processing time for the XGBoost-based LOFO model is 6943.35 seconds which is comparably higher than other tree-based learners. The training time for XGBoost classifier for XGBoost-LOFO selected features ranges from 0.50 seconds to 1 second. The testing time for the XGBoost model is 0.01 seconds which is very less. The feature processing time for LightGBM-LOFO is 1173.94. There has been only a slight increase in accuracy by comparing Light GBM-LOFO over XGBOOS\textsuperscript{T}T but there is a significant variation in the running time for the feature importance training process. Light GBM is almost 6 times faster than XGBOOST and is a much better approach when dealing with large datasets. Table 4 presents the efficiency of the proposed method by comparing the existing malware detection systems on DREBIN dataset.

### Table 3. Performance Comparison of the Android malware Feature Engineering and Classification with tree-based models based on Computational Time.

| Model      | Feature Processing Time (in seconds) | Number of features selected |
|------------|--------------------------------------|-----------------------------|
|            |                                      | Training Time (in seconds)  |
|            |                                      | 5  | 10 | 15 | 20 | 25 | 30 | 5  | 10 | 15 | 20 | 25 | 30 |
| Decision Tree | 868.52                           | 0  | 0.01| 0.01| 0.02| 0.02| 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| Random Forest | 3829.59                          | 0.27| 0.42| 0.45| 0.53| 0.58| 0.73| 0.03| 0.05| 0.06| 0.06| 0.07| 0.08|
| ExtraTrees  | 5136.77                           | 0.19| 0.28| 0.36| 0.42| 0.49| 0.53| 0.03| 0.04| 0.05| 0.05| 0.06| 0.06|
| GBM         | 4567.86                           | 0.22| 0.32| 0.42| 0.51| 0.61| 0.75| 0  | 0.01| 0.01| 0.01| 0.01| 0.01|
| Light GBM   | 1173.94                           | 0.18| 0.17| 0.18| 0.17| 0.2  | 0.19| 0.01| 0.02| 0.02| 0.02| 0.03| 0.03|
| AdaBoost    | 24999.25                          | 0.14| 0.18| 0.2  | 0.23| 0.31| 0.34| 0.02| 0.02| 0.03| 0.03| 0.03| 0.03|
| XGBoost     | 6943.35                           | 0.51| 0.64| 0.65| 0.76| 0.88| 0.89| 0.01| 0.01| 0.01| 0.01| 0.01| 0.02|
Table 4: Performance Comparison of the proposed work with other malware detection methods on Drebin dataset.

| Methods            | Acc (%) | Pr   | Re   | F1   |
|--------------------|---------|------|------|------|
| Arp et al. [2]     | 92.58   | 0.9186 | 0.9254 | 0.9220 |
| Cai et al. [11]    | 70.08   | 0.6884 | 0.7020 | 0.6951 |
| Pektas et al. [20] | 90.24   | 0.8972 | 0.9044 | 0.9008 |
| Taheri et al. [9]  | 94.31   | 0.9406 | 0.9428 | 0.9417 |
| Proposed Method    | 95.59   | 0.9485 | 0.9295 | 0.9389 |

5. Conclusion
An efficient Android malware detection and classification system was proposed by determining an optimal feature set based on the LOFO importance method. The efficiency of the proposed solution was calculated by tree-based model validation using different feature sets on the Android malware DREBIN dataset and the performance of tree models in Android malware detection was compared. The experimental results indicate the significance of the feature selection process which selects and discards the insignificant features and the performance improvements are assessed at every step. With minimal feature sets, the classification accuracy obtained is always substantially higher than with the complete feature set. The feature importance obtained from XGB based LOFO resulted in an optimal feature set and the XGB classifier showed an outperforming accuracy of 95.59% with a limited set of features.

Acknowledgment
The authors are grateful to VIT management for providing the Research Seed Grant (AY 2019-20) to execute this work.

References
[1] Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y 2012 “Andromaly”: A behavioral malware detection framework for android devices J. Intell. Inf. Syst. 38 161-190.
[2] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Mistele D, Rieck K 2014 Drebin: Effective and explainable detection of android malware in your pocket (NDSS vol 14) pp 23-26.
[3] Alam S, Qu Z, Riley R, Chen Y, Rastogi V 2017 DroidNative: Automating and optimizing detection of Android native code malware variants Comput. Secur. 65 230-246.
[4] Elish K O, Shu X, Yao D, Ryder B G, Jiang X 2015 Profiling user-trigger dependence for Android malware detection Comput. Secur. 49 255-273.
[5] Cai H 2020 Assessing and Improving Malware Detection Sustainability through App Evolution Studies ACM Trans. Softw. Eng. Methodol. 29 1-28.
[6] Roseline SA, Hari G, Geetha S, Krishnamurthy R 2019 Vision-Based Malware Detection and Classification Using Lightweight Deep Learning Paradigm Int. Conf. on Computer Vision and Image Processing (Singapore: Springer) pp 62-73.
[7] Garcia J, Hammad M, Malek S 2018 Lightweight, obfuscation-Resilient detection and family identification of android malware ACM Trans. Softw. Eng. Methodol. 26 1-29.

[8] Onwuzurike L, Mariconti E, Andriotis P, De Cristofaro E, Ross G, Stringhini G 2019 Mamadroid: Detecting android malware by building Markov chains of behavioral models (extended version) ACM Trans. Priv. Secur. 22 1-34.

[9] Taheri R, Ghahramani M, Javidan R, Shojafar M, Pooranian Z, Conti M 2020 Similarity-based Android malware detection using Hamming distance of static binary features Futur Gener Comput Syst. 105 pp 230-247.

[10] Roseline SA, Sasisri AD, Geetha S, Balasubramanian C 2019 Towards Efficient Malware Detection and Classification using Multilayered Random Forest Ensemble Technique Int. Carnahan Conf. on Security Technology (IEEE) pp 1-6.

[11] Cai H, Meng N, Ryder B, Yao D 2019 DroidCat: Effective android malware detection and categorization via app-level profiling IEEE Trans. Inf. Forensics Secur. 14 1455–1470.

[12] Quinlan JR 1996 Learning decision tree classifiers ACM Comput. Surv. 28 71-72.

[13] Breiman L 2001 Random forests Mach Learn. 45 5-32.

[14] Geurts P, Ernst D, Wehenkel L 2006 Extremely randomized trees Mach Learn 63 3-42.

[15] Friedman JH 2001 Greedy function approximation: A gradient boosting machine. Ann Stat. 29.

[16] Nielsen D 2016 Tree boosting with xgboost-why does xgboost win" every" machine learning competition? (Master's thesis, NTNU).

[17] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY 2017 LightGBM: A highly efficient gradient boosting decision tree Advances in Neural Information Processing Systems 30 pp 3146-3154.

[18] Liu J, Danait N, Hu S, Sengupta S 2013 A leave-one-feature-out wrapper method for feature selection in data classification 6th Int. Conf. on Biomedical Engineering and Informatics (IEEE) pp 656-660.

[19] Roseline SA, Geetha S, Kadry S, Nam Y 2020 Intelligent Vision-Based Malware Detection and Classification Using Deep Random Forest Paradigm IEEE Access 8 206303-24.

[20] Pektaş A, Acarman T. 2020 Learning to detect Android malware via opcode sequences Neurocomputing 396 pp 599-608.

[21] Hemalatha J, Roseline SA, Geetha S, Kadry S, Damaševičius R 2021 An Efficient DenseNet-Based Deep Learning Model for Malware Detection Entropy 23 344.