Android Malware Detection: An Empirical Investigation into Machine Learning Classifiers

Aaditya Raval
Human-Centered AI Lab, Computer Science Department
North Carolina Agricultural and Technical State University
Greensboro, NC, USA
ahraval@aggies.ncat.edu

Mohd Anwar
Human-Centered AI Lab, Computer Science Department
North Carolina Agricultural and Technical State University
Greensboro, NC, USA
manwar@ncat.edu

***Abstract*—Smartphones have become an all-in-one device due to their constant connectivity and ability to provide a wide range of functions. However, they can expose users to various forms of malware, posing significant threats to their privacy, security, and financial well-being. In this context, the detection and classification of Android malware have emerged as critical research areas in cybersecurity. Our study develops six distinct models/classifiers for detecting and classifying Android malware, leveraging the following machine-learning algorithms: MLP, logistic regression, random forest, SVM, XGBoost, and AdaBoost. Through a systematic evaluation process, we assess the efficacy of each model, highlighting their respective strengths and weaknesses. These findings not only contribute to the existing body of knowledge but also pave the way for future research and innovation in the field of Android security. Furthermore, we investigate the impact of data preprocessing and feature selection strategies on model performance and generalization capabilities. Our experimental results reveal that Random Forest (RF) and Extreme Gradient Boosting (XGBoost) classifiers outperformed others in classifying Android malware, showcasing performance of around 93% and 92%, respectively across accuracy, precision, recall, and f1-score. With an AUC of 0.93 for RF and 0.92 for XGBoost, these models can clearly distinguish between malware and benign samples with minimum misclassifications. Our findings shed light on the effectiveness of machine learning algorithms in combating Android malware and offer valuable insights into the most suitable models. Ultimately, this research study advances the understanding of Android malware detection and classification, providing a foundation for developing robust security solutions in the mobile computing landscape.**

Keywords—Android malware, malware detection, machine learning, ANOVA feature selection, malware analysis, mobile security, Android security, static analysis

I. Introduction

The internet has become an integral part of daily life for most people. The rapid growth of the internet provides a unique opportunity for cyber-criminals and other malicious actors to target a wide audience with various types of malware. Malware, short form for malicious software, disrupts computers, servers, smartphones, IoT devices, and networks to gain unauthorized access, leak private information, modify the existing information, and prevent access to the information. These malware behave differently depending on different objective they seek to achieve. For example, spyware gathers information about the victim's system, and ransomware encrypts the data on the victim's system and demands a monetary payment (ransom) to decrypt the encrypted files. Spyware, adware, ransomware, scareware, trojan horses, viruses, worms, rabbits, and wipers are a few examples of malware. Different malware can target smartphone applications to steal users' personal information, encrypt data, cause financial loss, and much more.

Statistics Portal [14] and StatCounter [15] show that Android and iOS cover 99% of the global smartphone market. Android and iOS are the most widely used mobile operating systems, with 71% Android users and 28% iOS users worldwide. Affordable cost, high customizations, open specifications, and better availability of third-party applications relative to competing alternatives are some key drivers for the popularity of Android smartphones. Due to open software specifications, users can select various application markets to download Android applications. The primary Android application distributor is the Google Play Store [16]. Some third-party Android application vendors are Amazon Appstore [18], Samsung Galaxy Store [19], F-Droid [20], APKMirror [21], APKPure [22], Aptoide [23], and Opera Mobile Store [24].

According to a recent Amazon Appstore statistics and trends 2024 [17] analysis report, 534,649 applications are currently available on Amazon Appstore with 123,993 active application publishers. Additionally, an average of 174 applications were released daily on Amazon Appstore. This trend shows a growing interest in third-party application vendors. Moreover, Microsoft Store and Amazon Appstore plan to form a partnership to offer Android applications on the popular Windows 11 operating system. The Android applications from both primary and third-party vendors can fall prey to attacks by cybercriminals, often due to the open-source nature of the Android operating system.

Amidst the exponential growth of the Android application ecosystem, the threat landscape is evolving at an alarming pace. Recent statistics reveal a staggering influx of Android malware, with approximately 12,000 new malicious applications emerging daily [25]. This flood of malware poses a significant and immediate threat to users and security professionals, underscoring the critical need for robust detection and mitigation strategies. Many researchers have used different

2835-5776/24/$31.00 ©2024 IEEE
DOI 10.1109/IRI62200.2024.00039

analysis techniques, such as static analysis, dynamic analysis, and hybrid approach, to analyze and detect malware samples. However, the quality and diversity of data from Android applications, both malware and benign samples, play a critical role in developing superior machine-learning models.

In this paper, we present six machine-learning models/classifiers for classifying and detecting Android malware using following machine learning algorithms: Multi-Layer Perceptron (MLP), Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), Extreme Gradient Boosting (XGBoost), and Adaptive Boosting (AdaBoost). By following a systematic method, we aim to address the challenges associated with Android malware detection and classification, highlight each model's strengths and weaknesses, and provide valuable insights into their suitability for Android security systems. Our research questions (RQ) and research contributions (RC) are as follows. RQ1: How effective are machine learning algorithms in classifying and detecting Android malware? RQ2: To what extent can data preprocessing and feature selection improve model performance and generalization ability? RC1: Developed machine learning models/classifiers for Android malware detection and classification. RC2: Provide a comprehensive analysis of each model's strengths and weaknesses. RC3: Analyze the impact of the most suitable data preprocessing and feature selection strategies on the high-dimensional Android application data.

The rest of the paper is organized as follows: Section 2 discusses the related work on Android malware detection and classification, providing insights into existing research efforts in the field. Section 3 introduces the methodology used in this paper, detailing the approach taken in developing and evaluating the proposed machine learning models. Section 4 presents the experimental results and discussion, analyzing the performance of each model and highlighting key findings. Finally, Section 5 concludes the paper, summarizing the study's contributions and outlining potential avenues for future research in Android malware detection and classification.

II. Related Work

Android malware detection and classification has experienced a surge in research due to the proliferation of mobile applications and the evolving threat landscape. In this section, we review the relevant literature, examining the various methodologies and techniques used to address the challenges of Android malware detection and classification. Our primary objective is to position our research within the broader context of existing literature, identifying key trends, innovations, and opportunities in Android malware mitigation strategies. Many existing studies rely on traditional static and dynamic analysis techniques for Android malware detection and classification.

Peiravian et al. [1] used a combination of permission and API calls to detect malicious Android applications using machine learning classifiers such as support vector machines, decision trees, and bagging predictors. Their experimental results showed that bagging predictors outperformed other models while achieving a good detection rate with a precision of up to 94.9% on the dataset containing 610 malware and 1250 benign samples. In contrast, Huang et al. [2] leveraged only permission data to detect more than 81% of malicious samples with the help of machine learning algorithms such as decision tree, adaboost, support vector machine, and naive bayes.

DREBIN [3] monitors static features of applications at runtime on smartphone devices. This method detected 94% of the malware samples at a false-positive rate of 1%, corresponding to one false alarm when installing one hundred applications. Takahashi et al. [4] analyzed the APK file structure and extracted permission requests, API calls, application clusters, and source descriptions. They utilized an SVM-based classifier to identify malware samples, which achieved 94.15% accuracy by removing the non-contributing features. The authors highlighted permission requests, API calls, and source descriptions were helpful in the detection process, while application clusters were not. Rahali et al. [5] proposed DIDroid, which utilizes image-based deep learning for Android malware classification. DIDroid primarily relies on image-based features of Android applications, processed using Convolutional Neural Networks (CNNs) for multi-class classification.

Like the above methods, our approach focuses on static features extracted directly from AndroidManifest files, focusing more on feature selection based on importance scores to reduce high dimensionality in the data. Additionally, while DIDroid [5] distinguishes between 12 malware categories, our research focuses on binary classification, distinguishing between benign and malicious Android applications. While both studies aim to improve Android malware detection, our research explores the efficacy of traditional machine-learning methods (MLP, LR, RF, SVM, XGBoost, and AdaBoost) in detecting and classifying Android malware.

On the other hand, dynamic analysis techniques concentrate on monitoring an application's runtime behavior, network traffic analysis, reverse engineering, and taint analysis. DroidScope [6] focuses on virtualization-based Android analysis by extracting Dalvik instruction traces for taint analysis; TaintDroid [7] performs dynamic taint analysis to detect suspicious behaviors during runtime. Similarly, Wang et al. [8] utilize semantic text analysis on network traffic to identify malicious Android applications based on NLP string analysis. Complementary to dynamic analysis techniques, our research emphasizes the importance of static features in enhancing the accuracy and efficacy of Android malware classification systems.

III. Methodology

In this section, we discuss the methodology employed in our research for classifying and detecting Android malware using machine learning techniques. Our methodology encompasses several steps: dataset selection, data preprocessing, feature selection, machine learning models, experimental setup, and evaluation metrics. Each step was carefully designed and executed to ensure the robustness and reliability of our research findings. By following a systematic approach, we aimed to address the complex challenges inherent in Android malware detection and provide valuable insights into the performance of different machine-learning models.

A. Dataset selection

For our research, we selected the CCCS-CIC-AndMal-2020 [10] dataset, a substantial and comprehensive resource provided by the Canadian Center for Cyber Security (CCCS) [11]. This

dataset, comprising 400,000 Android applications, is a testament to the credibility and comprehensiveness of our study. It includes 200,000 real-world malware samples collected by CCCS, labeled into 14 distinct malware categories using VirusTotal [9]. The dataset also incorporates 200,000 benign samples from the Androzoo dataset, which maintains a collection of over twenty-four million unique Android applications [12]. The inclusion of both malware and benign samples from reputable sources enhances the diversity and representativeness of our dataset, thereby strengthening the validity of our research findings.

The 14 malware categories in the CCCS-CIC-AndMal-2020 dataset encompass various malicious behaviors and threats encountered in the Android ecosystem. These categories include Adware, Backdoor, FileInfector, No_Category, Potentially Unwanted Apps (PUA), Ransomware, Riskware, Scareware, Trojan, Trojan-Banker, Trojan-Dropper, Trojan-SMS, Trojan-Spy, and Zero-Day [10]. Each category represents a distinct class of Android malware with unique characteristics and functionalities, enabling a granular analysis of the effectiveness of our machine-learning models. By leveraging this rich and diverse dataset, we aim to develop robust and reliable classifiers capable of accurately identifying and mitigating the evolving landscape of Android malware.

B. Data preprocessing

In the data preprocessing phase, we meticulously curated a balanced dataset for our binary classification experiments. We selected 20 samples from each of the 14 malware categories in the CCCS-CIC-AndMal-2020 dataset, resulting in 280 malware samples for our preliminary experiments. We plan to increase the number of samples significantly in future work. These samples were labeled "Malware" to signify their malicious nature. Simultaneously, we chose an equal number of 280 benign samples to maintain class balance. This sampling strategy, resulting in a dataset evenly distributed between malware and benign classes, ensures the reliability and fairness of our classification experiments. It fosters equitable training conditions for our machine-learning models and facilitates a fair assessment of their performance in detecting Android malware.

Expanding on the methodological framework established by Rahali et al. [5], our study embraced the resources provided by their research, specifically comma-separated files tailored for feature extraction from Android application package (APK) files. In alignment with their methodology, we relied on these pre-existing files as foundational elements for our feature selection process. These comma-separated files recorded static features such as activities, permissions, actions, categories, services, broadcast receivers-providers, metadata, and system features. Each feature was encoded into binary sequences within these files, adhering to the one-hot encoding format. Within these sequences, the presence of a feature was denoted by a value of one, while absence was represented by zero. This approach facilitated the transformation of feature attributes into structured feature vectors, laying the groundwork for subsequent stages of analysis and classification in our study.

In our experiment, we curated a balanced dataset containing 560 samples from the original dataset, ensuring an equal distribution of malware and benign samples. However, each sample in this dataset was characterized by a daunting 9,504 features. Recognizing the challenges posed by such high dimensionality and the potential for redundant or irrelevant features that can hinder model performance, we decided to use the Analysis of Variance (ANOVA) feature selection method. This approach enabled us to systematically identify and retain the most informative features while discarding those that were less relevant or potentially problematic for our classification models. By leveraging ANOVA-based feature selection, we aimed to streamline the dataset and focus on the subset of the most critical features for accurate and reliable classification of Android malware.

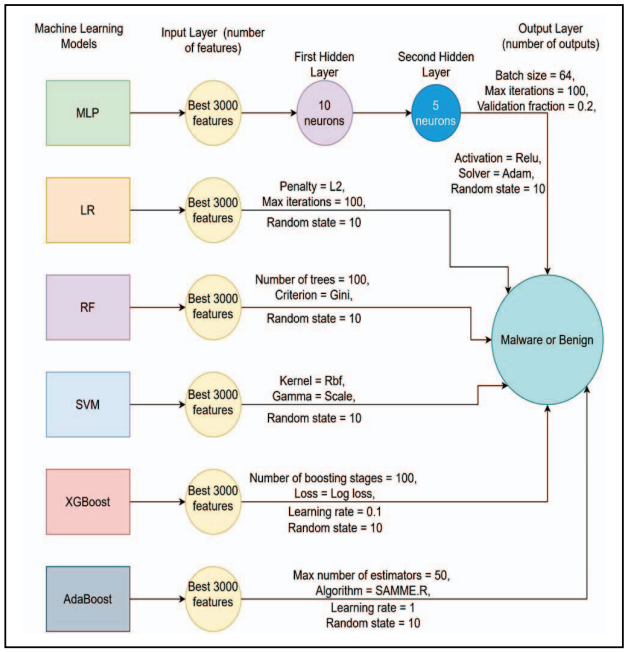

Fig. 1. *Architecture of ML models (MLP, LR, RF, SVM, XGBoost and AdaBoost) with hyperparameters.*

C. Feature selection

We opted for the ANOVA method in our feature selection process, which is best suited for this type of dataset where input is numerical variables, and output is categorical. ANOVA, a statistical technique commonly used in data analysis, assesses the variance between groups to determine the significance of differences. Leveraging ANOVA, we computed F-values to gauge the importance of individual features in contributing to the variability observed in the output categories. This method allowed us to discern which features exhibited the most substantial influence on the classification outcomes, enabling us to prioritize those with the most significant discriminatory power.

To streamline our feature selection process and enhance computational efficiency, we utilized the SelectKBest function [13]. This feature selection function identifies and retains the k highest scoring features based on their F-values computed through ANOVA. Given the dimensionality of our dataset and the potential redundancy or noise present in the feature space, selecting a subset of the most informative features is essential

for optimizing model performance and interpretability. We plotted the F-values on the graph to determine the most essential features. Based on the feature score and graph observations, we selected the top 3000 features with the highest scores using the SelectKBest function. We aimed to strike a balance between feature richness and computational tractability, ensuring that our models had a focused set of input variables capable of capturing the salient patterns within the data.

The ANOVA method and the SelectKBest function are employed in our feature selection process for model optimization and dimensionality reduction. To mitigate the risk of overfitting and enhance both models' interpretability and generalization capabilities, we leverage statistical analysis to discern the most relevant features while performing feature subset selection. Through this iterative feature selection process, we sought to distill the essential information from the dataset, empowering our models to make accurate and reliable predictions for Android malware classification.

D. Machine learning models

In the model selection process for our Android malware classification task, we carefully considered a range of machine-learning algorithms to identify those best suited to the complexities of the problem. The Multi-Layer Perceptron (MLP) neural network model was chosen for its ability to capture intricate patterns in data through multiple layers of interconnected neurons, making it well-suited for detecting subtle nuances indicative of malware behavior. Logistic Regression (LR) was selected for its simplicity and interpretability, offering insights into the probabilistic relationship between input features and the binary malware classification. Random Forest (RF) was deemed appropriate for its ensemble-based approach, which harnesses the collective wisdom of numerous decision trees to achieve robust classification performance, particularly effective in handling high-dimensional data like ours. Support Vector Machine (SVM) was included for its proficiency in separating data points using hyperplanes, making it particularly adept at delineating complex decision boundaries between malware and benign samples. We opted for Extreme Gradient Boosting (XGBoost) and Adaptive Boosting (AdaBoost) due to their ability to improve model performance by focusing on misclassified samples iteratively, thus enhancing classification accuracy and mitigating overfitting. Each model was selected based on its unique strengths and suitability for the Android malware classification task, aiming to provide a diverse ensemble of classifiers capable of capturing different aspects of the data and maximizing overall predictive performance. Figure 1 displays the architecture of each model.

E. Experimental setup

The experiments were conducted on a computing environment featuring an 11th-gen Intel(R) Core (TM) i7-1165G7 2.80GHz CPU, 12 GB RAM, and a 500 GB SSD disk, operating on a 64-bit Windows 11 Home Edition version 22H2 operating system. We used Python programming language and Scikit-learn library to develop our models/classifiers. Figure 1 outlines the hyperparameter configurations for each machine-learning model used in our experiments. For our binary classification experiments, we utilized our curated balanced dataset comprising 560 samples, each with 3000 features. This dataset was divided into 80% training set and 20% test set portions. We adopted a rigorous experimental approach leveraging 10-fold cross-validation on the training set to evaluate the selected models comprehensively. This methodology facilitates robustness and reliability by partitioning the dataset into 10 equally sized folds, using 9-folds for training and 1-fold for testing in each iteration. By incorporating 10-fold cross-validation, we aim to enhance the generalization capability of our models while providing a more stable and representative estimate of their performance. This iterative validation technique enables thorough assessment across multiple subsets of the data, effectively minimizing bias and variance in model evaluation and ensuring the reliability and robustness of our findings.

F. Evaluation metrics

In this research, we utilized a comprehensive set of evaluation metrics to assess the performance of our machine-learning models in classifying Android malware. These metrics included accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC). Accuracy measures the overall correctness of the model's predictions, while precision quantifies the proportion of true positive predictions among all positive predictions made by the model. Recall, also known as sensitivity, calculates the proportion of true positives correctly identified by the model out of all actual positives in the dataset. F1-score harmonizes precision and recall, providing a balanced assessment of a model's performance. Additionally, the AUC metric evaluates the model's ability to distinguish between positive and negative samples across various decision thresholds. Considering these diverse evaluation metrics, we aimed to comprehensively understand each model's performance and suitability for real-world deployment in Android malware classification tasks.

IV. Experimental Results And Discussion

In this section, we present a comprehensive evaluation of six distinct machine learning models: Multi-Layer Perceptron (MLP), Logistic Regression (LR) Classifier, Random Forest (RF), Support Vector Machine (SVM) Classifier, Extreme Gradient Boosting (XGBoost), and Adaptive Boosting (AdaBoost) for the classification and detection of Android malware. Rigorous experimentation on a diverse dataset containing benign and malicious samples has yielded significant findings. Each model was trained and evaluated using widely acceptable performance metrics such as accuracy, precision, recall, F1-score and area under the receiver operating characteristic curve (AUC). We also created a confusion matrix for each model to further understand classification performance. The results highlight each model's strengths and weaknesses and provide valuable insights into their suitability for real-world deployment in Android security systems.

Our investigation of Android malware classification involved evaluating the performance of various machine-learning models. As depicted in Table I, the Multi-Layer Perceptron (MLP) exhibited promising results, achieving a test accuracy of 90.17%. The MLP model demonstrated a robust ability to discern between benign and malicious applications, with a precision of 90.92%, recall of 90.18%, and an F1-score

of 90.20%. Similarly, the Logistic Regression (LR) Classifier attained a test accuracy of 88.39%, maintaining a balanced precision and recall of 89.56% and 88.39%, respectively, along with an F1-score of 88.40%. However, Random Forest (RF) emerged as the top performer, reporting an impressive test accuracy of 93.85% with a consistent precision, recall, and F1-score of around 93%. These results indicate the potential of Random Forest (RF) as a reliable model for malware detection, instilling optimism for its future applications.

TABLE I. CLASSIFICATION RESULTS

ML Classifier	**10-Fold CV (Training set)**	**Model Evaluation (Testing set)**				
	Accuracy (%)	***Accuracy (%)***	***Precision (%)***	***Recall (%)***	***F1-score (%)***	***AUC***
MLP	89.06	90.17	90.92	90.18	90.20	0.91
LR	87.05	88.39	89.56	88.39	88.40	0.89
RF	93.53	93.85	93.42	92.86	92.87	0.93
SVM	72.05	82.14	82.24	82.14	82.17	0.82
XGBoost	92.62	91.96	92.36	91.96	91.98	0.92
AdaBoost	92.18	88.39	89.12	88.39	88.41	0.89

In contrast, the Support Vector Machine (SVM) exhibited lower test accuracy at 82.14%. However, despite its lower overall accuracy, it maintained balanced precision and recall rates of around 82%, suggesting its capability to classify malware accurately, along with an F1-score of 82.17%. Furthermore, ensemble learning techniques such as XGBoost and AdaBoost demonstrated high test accuracy levels of 91.96% and 88.39%, respectively. XGBoost had a consistent precision, recall, and F1-score of around 92%. However, AdaBoost had a slightly lower precision of 89.12% and recall of 88.39%, with an F1-score of 88.41%. While XGBoost performed slightly better than AdaBoost in terms of accuracy, the evaluation of precision, recall, and F1-score metrics emphasizes the importance of considering multiple performance measures.

The confusion matrices provide a detailed breakdown of the classification performance of each machine learning model in distinguishing between benign and malicious Android applications. As we can see in Figure 2, starting with the Multi-Layer Perceptron (MLP) model, its confusion matrix indicates that out of a total of 112 test samples, 49 benign applications were correctly classified as benign (true negatives), while 52 malicious applications were accurately identified as malicious (true positives). However, the model misclassified 2 benign applications as malicious (false positives) and missed 9 malicious applications (false negatives), indicating room for improvement. Likewise, the Logistic Regression (LR) model demonstrates comparable performance, correctly classifying a substantial portion of samples while exhibiting similar misclassifications to MLP. These results suggest that while MLP and LR exhibit satisfactory classification accuracy, there is potential for refinement in minimizing false positive and false negative rates to enhance overall model efficacy.

In contrast, the Random Forest (RF) model's confusion matrix reveals impressive performance, with only 1 false positive and 7 false negatives demonstrating minimal misclassifications and a notably higher true positive rate. Random Forest (RF) is robust in distinguishing between benign and malicious applications, underscoring its efficacy in detecting Android malware. Similarly, XGBoost's confusion matrix highlights its impressive performance, with minimal false positives and negatives, matching closely with RF's classification performance. Still, we acknowledge that RF and XGBoost both missed 7 malicious applications.

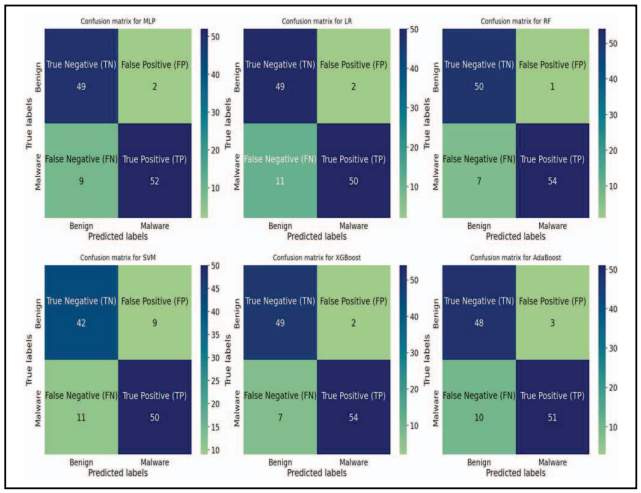

Fig. 2. *Confusion matrix for each ML model*

However, it is crucial to acknowledge that the performance of the Support Vector Machine (SVM) and AdaBoost models was comparatively inferior, having more misclassification than RF and XGBoost. Figure 3 shows the AUC-ROC curve for each model. The AUC results provide insights into the discriminative ability of each model in distinguishing between malicious and benign samples in the dataset. With an AUC of 0.93, Random Forest (RF) demonstrates the highest discriminative power among all models, indicating its effectiveness in classifying Android malware. Extreme Gradient Boosting (XGBoost) is closely behind, with an AUC of 0.92, further affirming its robust performance distinguishing between malware and benign samples. In contrast, the Support Vector Machine (SVM) has an AUC of 0.82, suggesting comparatively weaker discriminatory capabilities.

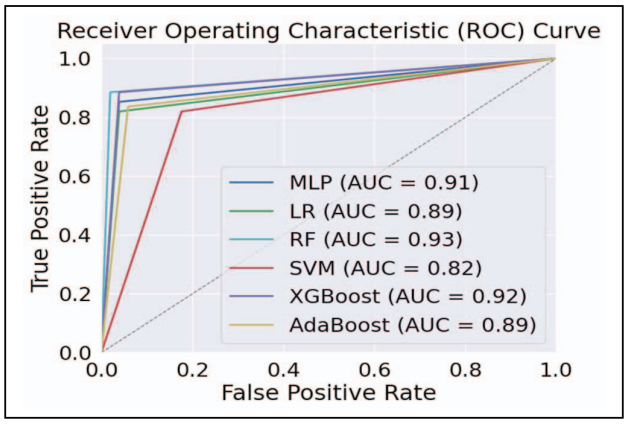

Fig. 3. *ROC curve for each ML model*

Overall, Random Forest (RF) and Extreme Gradient Boosting (XGBoost) emerge as the top-performing models with respect to AUC, showcasing their efficacy in Android malware classification tasks. Our results are close to or better than existing studies [1], [2], [4] in terms of accuracy, precision, and recall. However, our research focuses more on reducing sample misclassification, which we discussed based on confusion matrix results. These findings underscore the importance of considering the nuanced performance of various machine learning models in addressing the complex task of Android malware classification and highlight the potential of RF and XGBoost as promising candidates for deployment in real-world security systems.

V. Conclusion And Future Work

Our research addresses the pressing need for effective Android malware detection and classification mechanisms. By developing and rigorously evaluating six distinct machine-learning models/classifiers, we comprehensively analyze each model's strengths and weaknesses, shedding light on their performance and discriminative power. By incorporating our data preprocessing and feature selection strategies, we enhanced all the classifiers' efficacy and generalization capabilities, resulting in a notable 12 to 14% improvement in each model's overall performance. Based on the experimental results and minimum misclassifications, we determine that Random Forest (RF) and Extreme Gradient Boosting (XGBoost) are the most suitable candidates for deployment in Android security systems. We also acknowledge that RF and XGBoost models perform well in many machine-learning tasks. These contributions underscore the significance of our research in advancing the field of Android malware detection and classification in the cybersecurity domain.

In reflection on our research, it is imperative to recognize certain limitations that could inspire future investigations. Initially, our preliminary experiments were conducted on a relatively small dataset, limiting the scope of our findings. However, we aim to address this limitation by conducting similar experiments on a larger dataset containing 400,000 samples (200,000 benign and 200,000 malicious) in the future. Additionally, while we achieved promising results with our recommended model, further hyperparameter tuning could enhance its performance even more, contributing to more robust and accurate detection capabilities. To the best of our knowledge, we are using the most extensive dataset encompassing both malware and benign Android application samples; we recognize the value of exploring alternative datasets to ensure the comprehensive evaluation of our models. These identified limitations serve as valuable pathways for future research endeavors, paving the way for continued advancements in Android malware detection and classification.

References

[1] 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, 2013, pp. 300–305.

[2] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance Evaluation on Permission-Based Detection for Android Malware," in Advances in Intelligent Systems and Applications, vol. 2, Berlin, Heidelberg: Springer, 2013, pp. 111–120.

[3] D. Arp, M. Spreitzenbarth, M. H¨ubner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, USA, Feb. 2014.

[4] Takahashi, T., & Ban, T. (2019). Android application analysis using machine learning techniques. In AI in Cybersecurity (pp. 181-205). Springer, Cham.

[5] Rahali, A., Lashkari, A. H., Kaur, G., Taheri, L., Gagnon, F., & Massicotte, F. (2020, November). Didroid: Android malware classification and characterization using deep image learning. In Proceedings of the 2020 10th International Conference on Communication and Network Security (pp. 70-82).

[6] L. K. Yan and H. Yin, "DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in Proceedings of the 21st USENIX Conference on Security Symposium, 2012, pp. 29.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in Proceedings of the 9th USENIX conference on Operating systems design and implementation, vol. 57, no. 3, Mar. 2014., pp. 99–106.

[8] Wang, S., Yan, Q., Chen, Z., Yang, B., Zhao, C., & Conti, M. (2017). Detecting android malware leveraging text semantics of network flows. IEEE Transactions on Information Forensics and Security, 13(5), 1096-1109.

[9] VirusTotal. (2019). VirusTotal Website. Retrieved April 11, 2024, from https://www.virustotal.com/gui/home/upload

[10] Android Malware 2020. (2020). Retrieved April 11, 2024, from https://www.unb.ca/cic/datasets/andmal2020.html

[11] Canadian Centre for Cyber Security. (2020). Retrieved April 11, 2024, from https://cyber.gc.ca/en/

[12] Allix, K., Bissyandé, T. F., Klein, J., & Le Traon, Y. (2016, May). Androzoo: Collecting millions of android apps for the research community. In Proceedings of the 13th international conference on mining software repositories (pp. 468-471).

[13] scikit-learn. (n.d.). SelectKBest. Retrieved April 11, 2024, from sklearn.feature_selection.SelectKBest.html

[14] S. O'Dea, "The statistics portal," Feb. 2024. [Online]. Available: https://www.statista.com/statistics/266136/globalmarket-share-held-bysmartphone-operating-systems/

[15] Mobile Operating System Market Share Worldwide. StatCounter Global Stats. (n.d.). Retrieved April 11, 2024, from https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-202203

[16] Google Play Store. (n.d.). Retrieved April 11, 2024, from https://play.google.com/store/apps

[17] 42matters AG. (n.d.). Amazon Appstore Statistics and trends 2024. 42matters.Retrieved April 11, 2024, from https://42matters.com/amazon-appstore-statistics-and-trends

[18] Amazon. (n.d.). Mobile Apps. Retrieved April 11, 2024, from https://www.amazon.com/mobile-apps

[19] Samsung. (n.d.). Apps. Retrieved April 11, 2024, from https://galaxystore.samsung.com/apps

[20] F-Droid. (n.d.). Packages. Retrieved April 11, 2024, from https://f-droid.org/en/packages/

[21] APKMirror. (n.d.). Retrieved April 11, 2024, from https://www.apkmirror.com/

[22] APKPure. (n.d.). Retrieved April 11, 2024, from https://apkpure.com/app

[23] Aptoide. (n.d.). Retrieved April 11, 2024, from https://en.aptoide.com/

[24] Opera Mobile Store. (n.d.). Retrieved April 11, 2024, from https://apps.opera.com/

[25] G DATA Software AG. (2018, July 11). Cyber attacks on Android devices on the rise. Retrieved April 11, 2024, from https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise