AnLibsXAI: Android malware classification using library lists and explainable artificial intelligence with SHAP

DOI: https://doi.org/10.54654/isj.v2i22.1018

Nguyen Tan Cam*

***Abstract*— In this study, we use various features, including permission lists, API system calls, and library lists, to create a system named AnLibsXAI to classify Android malware. We experimented with six machine learning models such as Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR), Decision Tree (DT), K-Nearest Neighbors (KNN) and Multilayer Perceptron (MLP). Additionally, this study applies explainable artificial intelligence platforms to assess the importance and impact of features in Android malware classification models. The evaluation results on the CICMalDroid 2020 dataset show that SVM has the highest accuracy, reaching 96%. The findings of this study can be applied to future research related to Android malware classification.**

***Tóm tắt*— Trong nghiên cứu này, tác giả sử dụng các loại đặc trưng gồm danh sách thư viện, danh sách quyền hạn, lời gọi hàm hệ thống để xây dựng hệ thống phân loại mã độc Android có tên là AnLibsXAI. Tác giả đã thử sử dụng các mô hình học máy phổ biến như Support Vector Machine, Random Forest, Decision Tree, Logistic Regression, K-Nearest Neighbors và Multilayer Perceptron. Bên cạnh đó, nghiên cứu này cũng áp dụng các nền tảng trí tuệ nhân tạo có thể giải thích được để đánh giá tầm quan trọng cũng như ảnh hưởng của các đặc trưng trong các mô hình phân loại mã độc Android. Kết quả thử nghiệm khi sử dụng bộ dữ liệu thử nghiệm CICMalDroid 2020 cho thấy SVM có độ chính xác cao nhất, đạt 96%. Kết quả nghiên cứu này có ích trong việc sử dụng để hỗ trợ các nghiên cứu liên quan đến phân loại mã độc Android trong tương lai.**

***Keywords*— Android malware, malware classification, machine learning, explainable artificial intelligence, SVM.**

***Từ khóa*— Mã độc Android, phân loại mã độc, học máy, trí tuệ nhân tạo có thể giải thích, SVM.**

I. INTRODUCTION

As the Android operating system flourished, so did the prevalence and sophistication of Android malware. The widespread adoption of Android devices has made them lucrative targets for malicious actors seeking to exploit vulnerabilities for data theft, financial gain, or other nefarious purposes. The open nature of the Android ecosystem, while fostering innovation, also creates opportunities for malware to infiltrate devices through various means, such as malicious apps, phishing attacks, and system vulnerabilities.

The sheer volume of available apps on the Google Play Store, combined with the diversity of Android devices, presents a complex challenge for security measures. Consequently, the rise of Android malware underscores the importance of robust security solutions and the continuous adaptation of defense mechanisms to counter the evolving tactics employed by malicious entities in the dynamic landscape of mobile cybersecurity.

The pervasive use of Android devices in our daily lives has led to an unprecedented surge in mobile applications, making Android a prime target for malicious actors seeking to exploit vulnerabilities. The motivation for employing machine learning in Android malware classification is deeply rooted in the need for proactive and efficient defense mechanisms to safeguard user privacy, data integrity, and overall system security.

This manuscript was received on January 31, 2024. It was reviewed on May 25, 2024, revised on June 10, 2024 and accepted on June 28, 2024.

* Corresponding author

In essence, the motivation for employing machine learning in Android malware classification lies in its ability to provide a proactive, adaptable, and scalable defense mechanism against the evolving nature of mobile threats, ultimately ensuring the security and trustworthiness of Android devices in an interconnected world.

The Android operating system, being the predominant platform in the mobile ecosystem, faces an escalating threat from the increasing number of malicious applications. With the proliferation of Android malware, there is an urgent need for robust and effective classification models to identify and mitigate potential security risks. In response to this challenge, our research delves into a comprehensive analysis of Android malware detection, employing a diverse set of features, including permission lists, system call invocations, and library lists. The study leverages six machine learning models. The first one is SVM. The second one is RF. The third one is LR. The fourth one is DT. The next one is KNN. The last one is MLP.

As the threat landscape continues to evolve, understanding the intricate relationships between features and model outcomes becomes paramount. Therefore, our investigation goes beyond conventional classification models by incorporating explainable artificial intelligence (XAI) platforms such as LIME (Local Interpretable Model-agnostic Explanations) [1], SHAP (SHapley Additive exPlanations) [2], and LORE (Local Rule-based Explanations) [3],… These platforms contribute to elucidating the importance and impact of individual features in the classification process, enhancing the transparency and interpretability of the models.

Many studies focus on malware detection and applying machine learning for malware classification [4-12]. This study, conducted on the CICMalDroid 2020 dataset [13], aims to provide insights into the performance of various classification models in the context of Android malware detection. The outcomes of this study not only contribute to the advancement of Android malware detection techniques but also lay the groundwork for future research endeavors, including feature engineering and the application of explainable artificial intelligence in the realm of Android security.

The remaining content of the paper is structured as follows. Section 2 presents the related works. Section 3 details the proposed system. The evaluation of the proposed system is discussed in Section 4. Section 5 presents conclusions and future works.

II. Related works

Vinayakumar, et al. [5] demonstrated the effectiveness of Long Short-Term Memory (LSTM) networks in detecting and accurately classifying malicious applications by treating Android permissions as input to the LSTM network. The raw Android permissions are modeled at the permission level, employing the concept of a hierarchy model to assign numerical values for understanding the semantic meaning of permissions in Android and their correlations with other permissions.

Şahın, et al. [6] created two classifiers based on different rules. The system detecting malicious Android applications derived from the first rule is named LinRegDroid1, and the system detecting malicious Android applications derived from the second rule is named LinRegDroid2. The proposed system obtained is compared with SVM, KNN, and DT.

Angelo, et al. [7] proposed a system, named PermMaps, which combine Android permission information with the corresponding severity levels of those permissions to classify malicious Android applications. Their test results show that Random Forest (RF) classifier achieves an accuracy of 91%.

Zhang, et al. [8] proposed an Android malware detection system (AMDS) framework leveraging system call traces and machine learning techniques. By analyzing traces with N-gram and TF-IDF models, their system achieves early malware detection with an impressive average accuracy of 99.34%. The study includes the training of six machine learning algorithms and demonstrates the AMDS's real-time deployment.

Vinayaka, et al. [9] proposed a model for detecting Android malware using Graph Convolutional Networks (GCN) based on the Function Call Graph (FCG) which records the caller relationships and the relationships of methods within the Android Package Kit (APK) as a directed graph. Each node in the FCG is assigned a feature vector representing its characteristics. A series of experiments were conducted by varying the GCN algorithm, node features, and the number of GCN layers in the model, followed by evaluations. When the GCN considers the number of nodes in the FCG, the data is balanced by using a new technique to distribute the number of nodes between malicious and benign APKs equally. The results of these experiments achieved an accuracy of 92.29% with an F1 score of 92.23%, indicating that GCN has the capability to detect Android malware.

Wu, et al. [10] proposed DeepCatra. It is a cutting-edge multi-view learning solution for Android malware detection, employing a bidirectional LSTM (BiLSTM) and a graph neural network (GNN) as subnets. The model leverages features extracted from call traces leading to critical APIs. By constructing call graphs, computing call traces, and utilizing temporal opcode features for the BiLSTM subnet, along with flow graph features for the GNN subnet, their system showcases substantial improvements over state-of-the-art approaches. Its effectiveness is evident through notable enhancements, such as 2.7% - 14.6% on the F1 measure.

Kim, et al. [11] proposed MAPAS, a model for AMD with relatively high accuracy and adaptable resource consumption. MAPAS analyzes the behavior of malware based on their API call graphs using a Convolutional Neural Network (CNN). However, this system does not use the classification model by using CNN; it utilizes Convolutional Neural Network to discover API call graph of Android malware. To efficiently detect Android malware, the system employs a lightweight classifier to compute the similarity between the API call graph used for malicious behavior and the API call graph of apps awaiting classification. MAPAS achieves high prediction accuracy (91.27%) compared to MaMaDroid (84.99%). In the other study, Ding et al. used CNN for Android malware detection [14] by using bytecode image.

Mat, et al. [12] proposed an AMD system by using permission as features with Bayesian classification. They collected and evaluated 10,000 samples from the AndroZoo [15] and Drebin [16] datasets. Subsequently, tests were conducted by using chi-square and information gain. The detection rate for permission features achieved the highest accuracy at 91.1%.

Martin et al. [17] proposed an Android malware detection system using CNN and XAI. They used LIME [1] to explain the Android malware classification model. They utilized opcode to convert into images before using CNN to classify these images. The use of opcode as a feature is susceptible to obfuscation techniques. This research needs to be extended by using other XAI frameworks such as SHAP, as well as using other types of features like permissions, libraries, etc.

Galli et al. [18] proposed AI-based behavioral malware detection systems. They analyzed API calls to detect Android malware. They used four XAI models. This research needs to be tested with more types of features such as permissions and library lists.

Thus, there are many studies using machine learning and XAI to detect Android malware. Some studies use XAI [17][18] while others do not. This research inherits the methods of applying XAI in Android malware detection from related studies. We use the same approach to explain different feature sets. Specifically, in this study, we focus on permissions, API system calls, and library lists. In this study, we apply explainable artificial intelligence (XAI) to explain the contribution of features on the Android malware classification model, thereby proposing feature selection methods in the future.

III. PROPOSED SYSTEM

We propose the Android malware classification system, named AnLibsXAI. The proposed system has a structure as shown in Figure 1. The proposed system in this study consists of five main modules, including Feature

Extractor module, Feature Selector module, Classifiers module, and Models Explainer module.

A. Feature extractor

Feature extractor module is used to extract features for the purpose of training machine learning models. We utilize three types of features, including permissions lists, API calls, and libraries lists. Among them, permissions and API calls are extracted using AndroPyTool [19]. Libraries lists are extracted using a tool developed by ourselves.

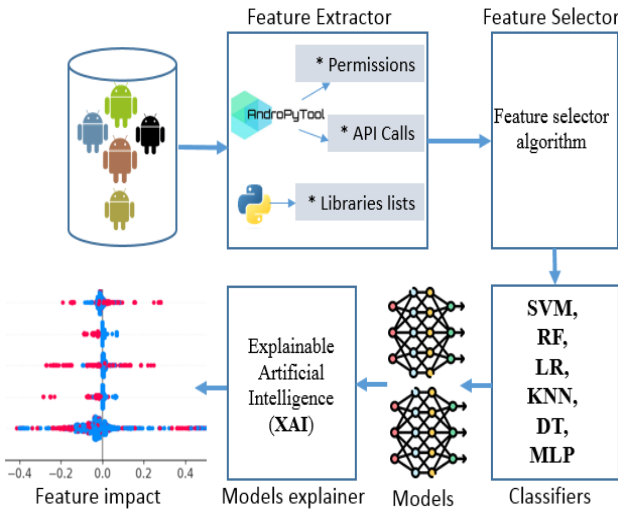

Figure 1. The proposed system architecture

The steps in the process of extracting permissions and API calls are as follows:

Step 1: APKs filtering: AndroPyTool categorizes the input files to determine whether they are APK files.

Step 2: VirusTotal analysis: The use of VirusTotal to scan and analyze the initial samples to assess their reliability and safety. VirusTotal typically provides detailed results on the number of antivirus tools detecting whether the sample is malicious.

Step 3: Dataset partition: AndroPyTool divides the dataset into different portions: benign (safe) and malware.

Step 4: FlowDroid [20] Execution: Analyzing the Android source code to identify data flows between different components in the application.

Step 5: FlowDroid output processing: AndroPyTool can extract information about data flows between components in the application from the results of FlowDroid, including both the source and destination of the data, as well as related components.

Step 6: Features extraction and processing: AndroPyTool can gather permissions that an application requests when installed on an Android device. These permissions are often used to determine the application's impact on the system and user data. AndroPyTool can check and collect information about system function calls that the application performs during runtime. This provides insight into the application's behavior and can be used to detect malicious activities.

Code 1. Libraries extractor algorithm

Algorithm 1 Extract Libraries Lists
Input: apkFiles: The list of APK files
Output: featureData: Libraries lists

```
procedure COLLECT-LIBRARY-LISTS(apkFiles)
    libraryList ←
    for each apkFile in apkFiles do
        extractedLibraries ← EXTRACT-LIBRARIES-FROM-APK(apkFile)
        libraryList ← libraryList ∪ extractedLibraries
    end for
    return list(libraryList)
end procedure

procedure SELECT-LIBRARIES-AS-FEATURES(libraryList)
    selectedLibraries ←
    for each library in libraryList do
        totalOccurrence ← CALCULATE-TOTAL-OCCURRENCE(library, allAPKFiles)
        if totalOccurrence >20 then
            selectedLibraries ← selectedLibraries ∪ library
        end if
    end for
    return list(selectedLibraries)
end procedure

procedure EXTRACT-FEATURE-DATA(selectedLibraries, apkFiles)
    featureData ←
    for each apkFile in apkFiles do
        extractedLibraries ← EXTRACT-LIBRARIES-FROM-APK(apkFile)
        featureValues ← [1 if lib in extractedLibraries else 0 for lib in selectedLibraries]
        featureData ← featureData ∪ featureValues
    end for
    return featureData
end procedure
```

TABLE 1. SOM EXAMPLES OF FEATURE REPRESENTATION

WRITE_SYNC_SETTINGS	launcher.permission.WRITE_SETTINGS	...	libgdx.so	libstlport_shared.so	libhunt.so	libzpay.so	libdnlocal.so
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
...	...	...	...	...	...	...	...
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0
0	0	...	0	0	0	0	0

The process of extracting libraries lists is carried out according to Algorithm 1 (Code 1). The algorithm aims to extract libraries lists from a set of APK files, selecting a subset of libraries as features based on their total occurrence across the dataset. The process involves collecting the library names from the APK files, filtering the libraries based on predefined conditions (e.g., total occurrence greater than 20), and then extracting feature data by evaluating the presence or absence of selected libraries in each APK file.

The algorithm employs functions for extracting libraries from APK files, calculating the total occurrence of each library, and returns a list of selected libraries as features and corresponding feature data. The ultimate goal is to create a meaningful representation of libraries present in Android applications for further analysis, such as in the context of machine learning model training. Table 1 shows some examples of feature representation.

B. Feature selector

The feature selection process is crucial for training models efficiently, focusing on identifying and retaining the most important attributes while reducing unnecessary ones to enhance model performance. By attentively considering the most significant attributes and reducing the number of irrelevant features, this process contributes to improving the efficiency and effectiveness of the model.

Dimensionality reduction is achieved by selecting features that appear between 10% and 95% to identify common features. The steps of the feature selection process are presented in Algorithm 2 (Code 2).

Code 2. Feature selector algorithm

Algorithm 2 Feature Selector

Input: Origin dataset (inputCSV), lowThreshold, highThreshold

Output: Modified dataset saved in outputCSV

```
procedure FEATURE-SELECTOR(inputCSV, outputCSV, lowThreshold, highThreshold)
    Step 1: Load dataset
    data ← READ-CSV(inputCSV)
    Step 2: Calculate variances
    variances ← CALCULATE-VARIANCES(data)
    Step 3: Identify low and high variance columns
    lowVarianceCols ← FIND-LOW-VARIANCE-COLUMNS(variances, lowThreshold)
    highVarianceCols ← FIND-HIGH-VARIANCE-COLUMNS(variances, highThreshold)
    Step 4: Remove low variance columns
    REMOVE-COLUMNS(data, lowVarianceCols)
    Step 5: Remove high variance columns
    REMOVE-COLUMNS(data, highVarianceCols)
    Step 6: Save modified dataset
    WRITE-CSV(data, outputCSV)
end procedure

function READ-CSV(filename)
    end function

function CALCULATE-VARIANCES(data)
    end function

function FIND-LOW-VARIANCE-COLUMNS(variances, threshold)
    end function

function FIND-HIGH-VARIANCE-COLUMNS(variances, threshold)
    end function

function REMOVE-COLUMNS(data, columns)
    end function

function WRITE-CSV(data, filename)
    end function
```

This algorithm begins by loading the dataset from an input CSV file and then calculates the variance for each column. The columns with variance below a specified low threshold or above a high threshold are identified. The algorithm proceeds to remove the low variance columns, followed by the high variance columns. Finally, the modified dataset, now devoid of columns with extreme variances, is saved to an output CSV file. This process ensures that only the most relevant features are retained, thereby improving the performance of machine learning models.

C. Classifiers

In this study, we employ six classification models. The first one is SVM (Support Vector Machine). The second one is RF (Random Forest). The third one is LR (Logistic Regression). The next one is DT (Decision Tree). The two last ones are KNN and MLP.

SVM is a supervised machine learning algorithm. It can be used for regression and classification tasks. This algorithm works by finding the hyperplane that best separates different classes in the feature space.

RF stands as an ensemble learning technique applied in classification and regression tasks. RF constructs numerous decision trees and provides the mode of classes for classification or the mean prediction for regression, aggregating the results from the individual trees in the training process.

LR is a classification algorithm. This algorithm models the probability of a binary outcome using a logistic function. It is widely used for binary classification problems.

KNN serves as a straightforward yet efficient algorithm applicable to both classification and regression tasks. In the KNN approach, the classification of an object is determined by a majority vote from its k nearest neighbors, relying on a specified distance metric like Euclidean distance.

DT embody a tree-like model, wherein internal nodes symbolize features or attributes. In this model, branches denote decision rules. In this model, leaf nodes signify the outcome. This model finds utility in tasks involving both classification and regression.

The MLP is a key architecture in artificial neural networks, featuring layers of interconnected nodes, including input, hidden, and output layers. It employs weights and activation functions to learn complex patterns and relationships in data. Through backpropagation during training, the network adjusts weights to enhance prediction accuracy. MLPs are versatile, commonly used for tasks like classification and regression due to their ability to capture intricate data patterns and handle non-linear relationships.

D. Models explainer

Explainable Artificial Intelligence (XAI) techniques serve the purpose of providing transparency and interpretability to machine learning models, including those used for Android malware classification. There are some XAI techniques such as LIME (Local Interpretable Model-agnostic Explanations) [1], SHAP (SHapley Additive exPlanations) [2], and LORE (Local Rule-based Explanations) [3]. XAI methods help make the machine learning models more understandable. For Android malware classification, knowing how a model arrives at a particular classification decision is crucial for users, developers, and security analysts. XAI techniques assist in validating the reliability and trustworthiness of the machine learning model. Users and stakeholders can gain confidence in the model's predictions if they can understand the features or factors contributing to a particular classification. Techniques like SHAP and LIME can highlight the importance of specific features in the decision-making process. This information is valuable for understanding the impact of different features on the model's predictions in the context of Android malware.

In this study, we use SHAP to explain Android malware classification models. This approach allows us to identify the importance of features and their impact within the classification models. The Shapley value for a feature is calculated using the following formula [21]:

$$\phi_i(f) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|! \cdot (|N| - |S| - 1)!}{|N|!} \cdot (f(S \cup \{i\}) - f(S)) \quad (1)$$

Where:

• $\emptyset_i(f)$ is the Shapley value for feature *i*.

• *N* is the set of all features.

• *S* is a subset of features not including *i*.

• |*S*| is the size of subset S.

• *f (S)* is the model output with the feature subset *S*.

IV. Evaluation

A. Dataset

This study uses the CICMalDroid 2020 dataset [13]. This dataset consists of 5 classes, including Adware, SMS malware, Banking malware, Riskware, and Benign. Figure 2 illustrates the distribution of the number of samples within each class. In this study, the dataset is divided into two sets. The first one is the training set, constitutes 70%. The second one is testing set, constitutes 30%.

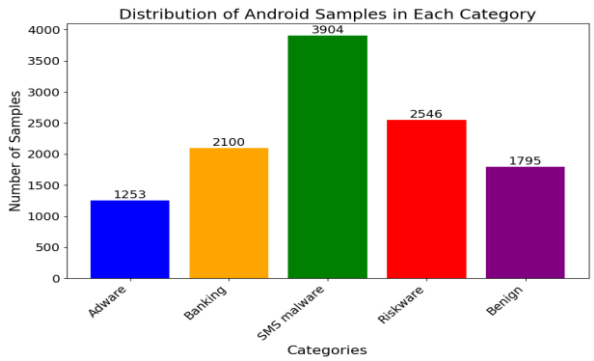

Figure 2. The distribution of the number of samples within each class

B. Evaluation metrics

In this study, we use the four main evaluation metrics including Recall, Precision, Accuracy, and F1-Score. Precision measures the proportion of accurately predicted positive instances relative to the total predicted positive instances. This metric evaluates the precision of positive predictions generated by the model.

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

where *FP* is the number of false positives, and *TP* is the number of true positives.

Recall measures the ability of the model to capture all the positive instances.

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

where *TP* is the number of true positives and *FN* is the number of false negatives.

The F1-Score represents the harmonic mean of precision and recall, offering a balanced assessment that considers both precision and recall. This metric is particularly useful in situations where there exists an imbalance between the quantities of positive and negative instances.

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision+Recall} \quad (4)$$

Accuracy is used as the ratio of correctly predicted observations to the total number of observations.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (5)$$

where *TP* is true positives, *FP* is false positives, *FN* is false negatives, and *TN* is true negatives.

B. Evaluation of classification models

Figure 3 illustrates the confusion matrix of the RF algorithm. The high accuracy rate for class 0 (Benign) indicates that the majority of benign APKs are correctly predicted. However, there is a relatively high misclassification rate for class 4 (SMS), where the model makes inaccurate predictions. In such cases, the model's predictions are relatively poor, and there are many errors.

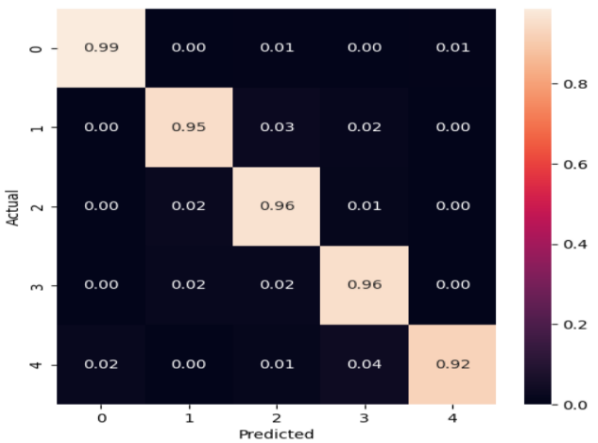

Figure 3. The confusion matrix of the RF algorithm

Figure 4 shows the confusion matrix of the LR algorithm. In this model, the accuracy rate for

class 0 (Benign) is the highest, while the accuracy rate for label 4 (SMS) is the lowest. Incorrect classifications for class 3 (Banking) and 4 lead to confusion, as labels 3 and 4 are often misclassified into other labels.

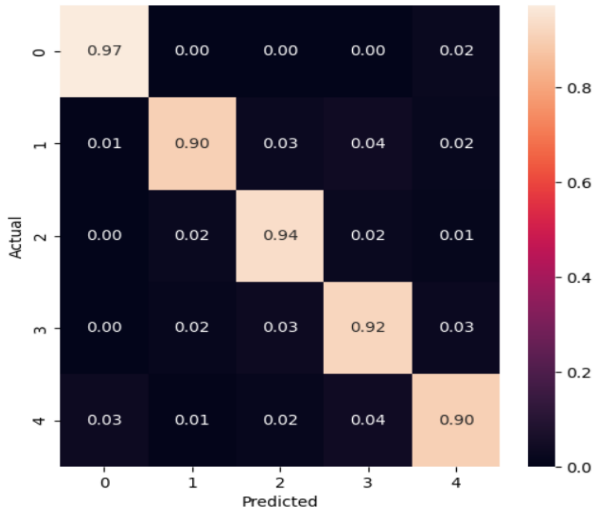

Figure 4. The confusion matrix of the LR algorithm

Figure 5 displays the confusion matrix of KNN (k = 3). This matrix indicates misclassifications for all labels (0, 1, 2, 3, 4). The accuracy rate for classifying APKs into their respective labels is not high.

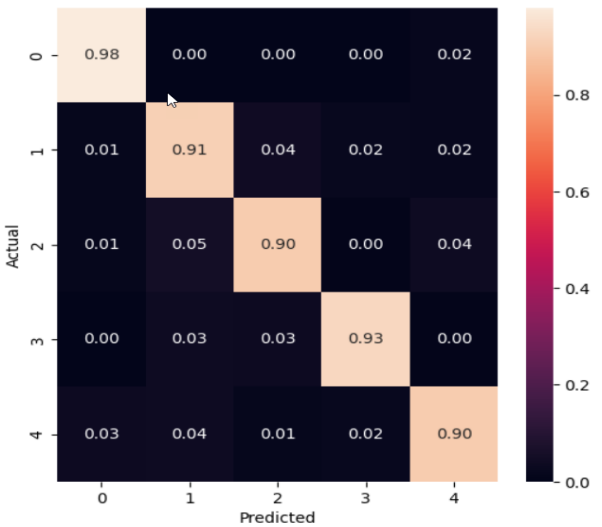

Figure 5. The confusion matrix of the KNN algorithm

Figure 6 presents the confusion matrix of the decision tree model. In this model, most of the data samples are misclassified, lacking accuracy. Specifically, the data with label 4 (SMS) is misclassified the most, often confused with other labels, and the highest misclassification occurs with label 1, with an error rate of 8%.

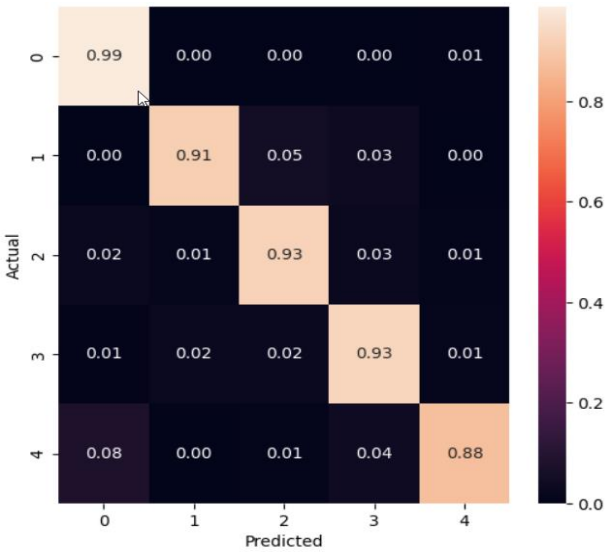

Figure 6. The confusion matrix of the DT algorithm

Figure 7 presents the confusion matrix of the SVM model. The accuracy rate for correctly predicting label 0 is the highest, while the accuracy rate for correctly predicting label 4 is the lowest (though higher than the results of other model experiments). The misclassification rate between labels is relatively low and acceptable.

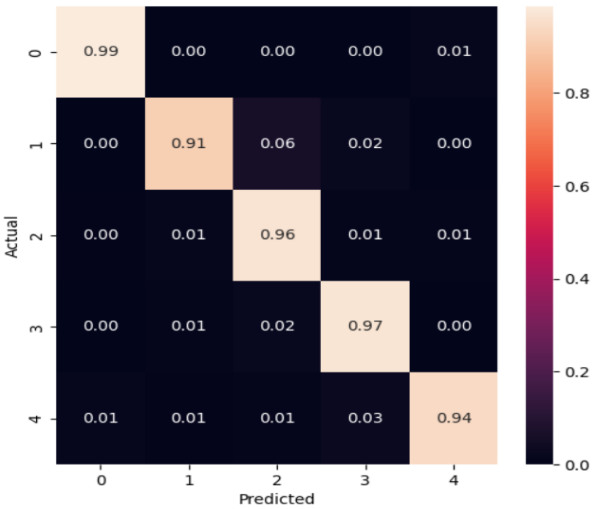

Figure 7. The confusion matrix of the SVM algorithm

Figure 8 displays the confusion matrix of the MLP. The accuracy rate for predicting data in each label is relatively high and consistent. However, there is still a phenomenon of misclassifying label 4 into other labels.

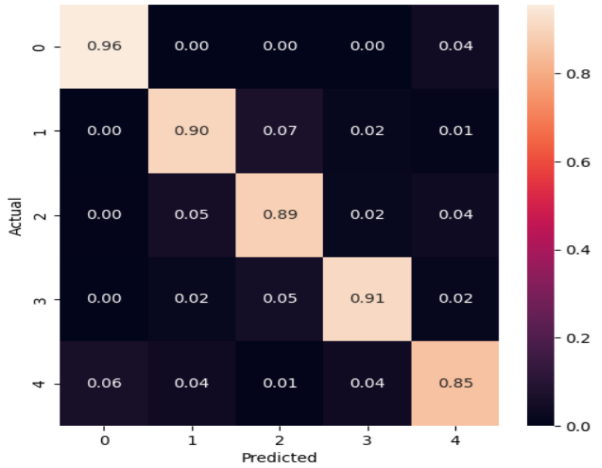

Figure 8. The confusion matrix of the MLP algorithm

Table 2 presents the evaluation results of the classification models used in this study. The results indicate that SVM has the highest accuracy.

TABLE 2. THE EVALUATION RESULTS OF THE CLASSIFICATION MODELS

Models	**Accuracy**	**Precision**	**Recall**	**F1-Score**
RF	0.95746	0.95658	0.95393	0.95500
LR	0.93148	0.92849	0.92724	0.92782
KNN	0.92656	0.91889	0.92251	0.92020
DT	0.93283	0.93373	0.92741	0.93005
SVM	**0.96014**	**0.95924**	**0.95498**	**0.95693**
MLP	0.90640	0.89882	0.90235	0.90034

In this study, we use SHAP to explain the importance of features for each classification model. Figure 9, Figure 10, Figure 11, and Figure 12 illustrate the results determining the importance of features to these models. By using the formula (1) to calculate the shapley value of each feature, the results indicate that "SEND_SMS" is the most important feature.

The results were obtained using the summary_plot function of SHAP, illustrating the influence of malicious features on the output of each model.

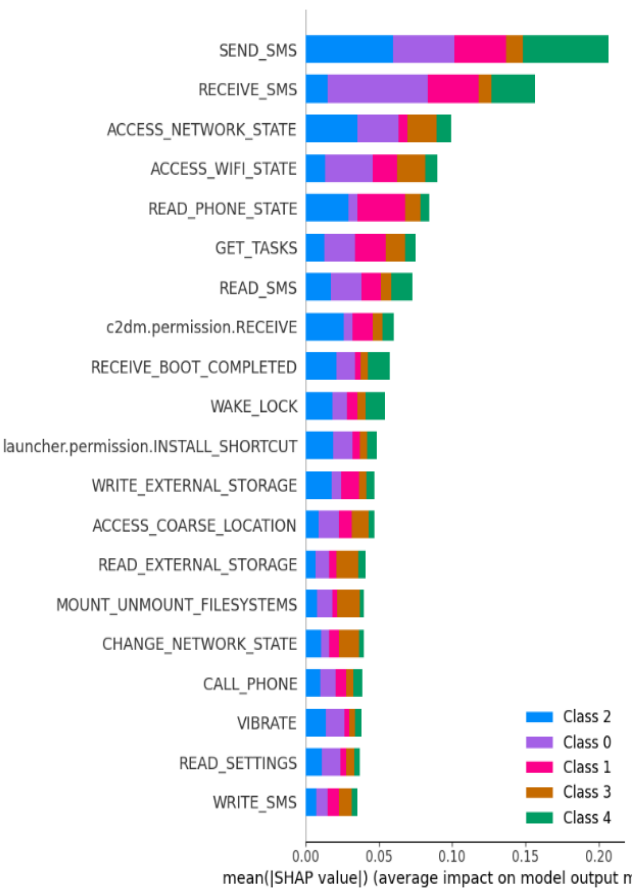

Figure 9. The results explain the RF model using SHAP

Observing the four plots, it is evident that while the influence of features differs among the Random Forest, Logistic Regression, and Decision Tree models, the two features that have the most impact on the output of these three models are "SEND_SMS" and "RECEIVE_SMS". In contrast, for the SVM model, the output is most affected by the features "RECEIVE_SMS" and "RECEIVE_BOOT_COMPLETED".

"RECEIVE_SMS" stands out as the most influential feature on the output of all four models when considering all five classes. However, when examining each class individually, there may be variations.

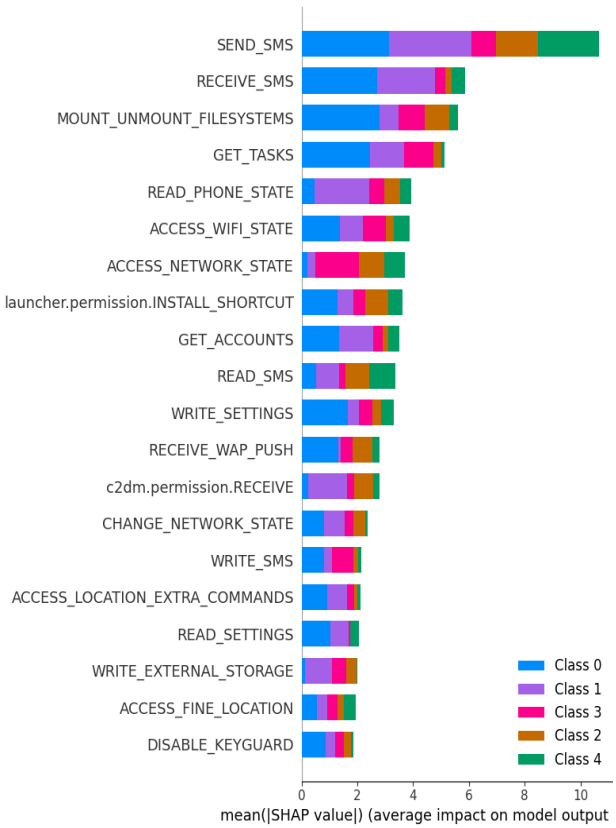

Figure 10. The results explain the LR model using SHAP

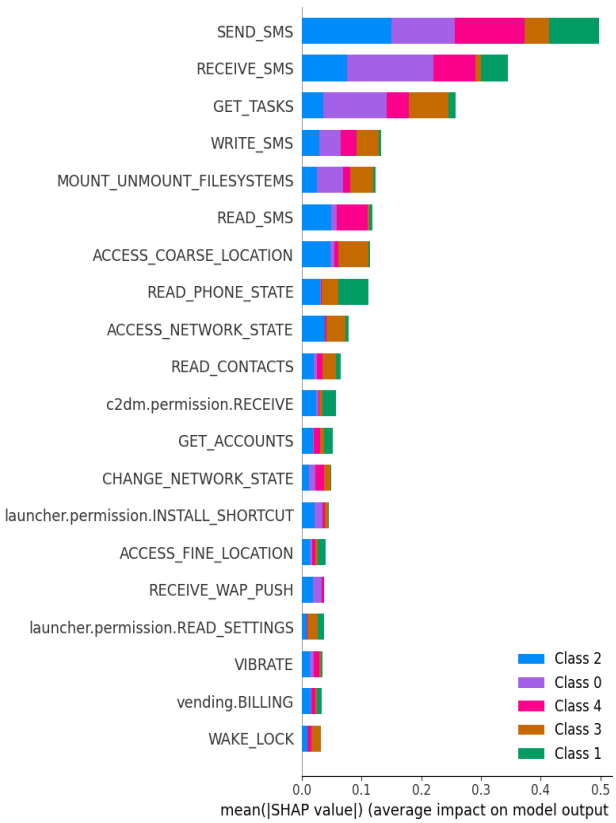

Figure 11. The results explain the DT model using SHAP

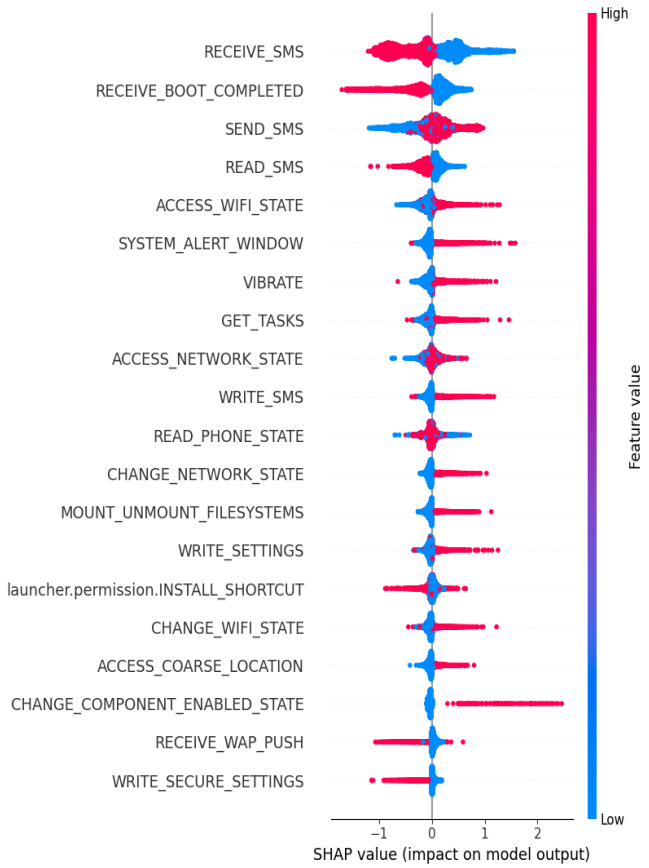

Figure 12. The results explain the SVM model using SHAP

IV. Conclusion

In conclusion, this study addresses the escalating issue of Android malware by employing a comprehensive approach that involves utilizing diverse features, including permission lists, API system calls, and library lists. The evaluation of various machine learning models, such as SVM, RF, LR, KNN, DT, and MLP, reveals that SVM attains the highest accuracy of 0.96 on the CICMalDroid 2020 dataset. Furthermore, the incorporation of interpretable artificial intelligence platforms facilitates a deeper understanding of the importance and influence of features within the Android malware classification models.

In the future, we can apply the findings of this research to conduct more in-depth studies in feature engineering. This would be beneficial in the development of Android malware classification models with fewer features but requiring fewer resources. These models could be deployed on real-world mobile phones.

ACKNOWLEDGMENT

This research was supported by the VNUHCM-University of Information Technology's Scientific Research Support Fund.

REFERENCES

[1] B. Wang, W. Pei, B. Xue, and M. Zhang, "A multi-objective genetic algorithm to evolving local interpretable model-agnostic explanations for deep neural networks in image classification," *IEEE Transactions on Evolutionary Computation*, 2022.

[2] S. Lundberg. (2023, October, 10). *SHapley Additive exPlanations*. Available: https://shap.readthedocs.io/en/latest/

[3] D. Macha, M. Kozielski, Ł. Wróbel, and M. Sikora, "RuleXAI—A package for rule-based explanations of machine learning model," *SoftwareX*, vol. 20, p. 101209, 2022.

[4] Toan, N. N. ., Dung, L. T., & Thang, D. Q. (2022). Static Feature Selection for IoT Malware Detection. Journal of Science and Technology on Information Security, 1(15), 74-84. https://doi.org/10.54654/isj.v1i15.844

[5] R. Vinayakumar, K. Soman, and P. Poornachandran, "Deep android malware detection and classification," in *2017 International conference on advances in computing, communications and informatics (ICACCI)*, 2017, pp. 1677-1683: IEEE.

[6] D. Ö. Şahın, S. Akleylek, and E. Kiliç, "LinRegDroid: Detection of Android malware using multiple linear regression models-based classifiers," *IEEE Access*, vol. 10, pp. 14246-14259, 2022.

[7] G. D'Angelo, F. Palmieri, and A. Robustelli, "A federated approach to Android malware classification through Perm-Maps," *Cluster Computing*, vol. 25, no. 4, pp. 2487-2500, 2022.

[8] X. Zhang *et al.*, "An early detection of android malware using system calls based machine learning model," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1-9.

[9] K. Vinayaka and C. Jaidhar, "Android malware detection using function call graph with graph convolutional networks," in *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, 2021, pp. 279-287: IEEE.

[10] Y. Wu, J. Shi, P. Wang, D. Zeng, and C. Sun, "DeepCatra: Learning flow-and graph-based behaviours for Android malware detection," *IET Information Security*, vol. 17, no. 1, pp. 118-130, 2023.

[11] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, "MAPAS: a practical deep learning-based android malware detection system," *International Journal of Information Security*, vol. 21, no. 4, pp. 725-738, 2022.

[12] S. R. T. Mat, M. F. Ab Razak, M. N. M. Kahar, J. M. Arif, and A. Firdaus, "A Bayesian probability model for Android malware detection," *ICT Express*, vol. 8, no. 3, pp. 424-431, 2022.

[13] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of network and systems management*, vol. 30, pp. 1-34, 2022.

[14] Y. Ding, X. Zhang, J. Hu, and W. Xu, "Android malware detection method based on bytecode image," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 5, pp. 6401-6410, 2023.

[15] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*, 2016, pp. 468-471: IEEE.

[16] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," 2014.

[17] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, "Towards explainable CNNs for Android malware detection," *Procedia Computer Science*, vol. 184, pp. 959-965, 2021.

[18] A. Galli, V. La Gatta, V. Moscato, M. Postiglione, and G. Sperlì, "Explainability in AI-based behavioral malware detection systems," *Computers & Security*, vol. 141, p. 103842, 2024.

[19] A. Martín, R. Lara-Cabrera, and D. Camacho, "A new tool for static and dynamic Android malware analysis," in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)*, 2018, pp. 509-516: World Scientific.

[20] D. Boxler and K. R. Walcott, "Static Taint Analysis Tools to Detect Information Flows," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2018, pp. 46-52: The Steering

Committee of The World Congress in Computer Science, Computer ….

[21] C. Molnar, "Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.", 2020, [Online]. Available: https://christophm.github.io/interpretable-ml-book/. [Accessed: Dec. 10, 2023].

ABOUT THE AUTHOR

Nguyen Tan Cam

Workplace: (1) University of Information Technology, Ho Chi Minh City, Vietnam. (2) Vietnam National University, Ho Chi Minh City, Vietnam.

Email: camnt@uit.edu.vn

Education: His bachelor degree in Information Technology from the University of Sciences, Vietnam National University Ho Chi Minh City in 2006. He received his Master degree in Information Systems from this university in 2010. He received his PhD degree of Information Technology from University of Information Technology, Vietnam National University Ho Chi Minh City (UIT VNU-HCM) in 2021. He is a lecturer at UIT VNU-HCM. He is also an active reviewer for Computer & Security journal (ISSN: 0167-4048).

Recent research direction: mobile security, network security, machine learning, and deep learning.

Tên tác giả: **Nguyễn Tấn Cầm**

Cơ quan công tác: (1) Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh, Việt Nam; (2) Đại học Quốc gia Thành phố Hồ Chí Minh, Việt Nam.

Email: camnt@uit.edu.vn

Quá trình đào tạo: Tốt nghiệp Cử nhân Công nghệ thông tin tại Trường Đại học Khoa học tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh năm 2006. Tốt nghiệp Thạc sĩ Hệ thống thông tin tại Trường Đại học Khoa học tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh năm 2010. Tốt nghiệp Tiến sĩ Công nghệ thông tin tại Trường Đại học Công nghệ thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh năm 2021. Ông là giảng viên của Trường Đại học Công nghệ thông tin, Đại học Quốc gia TP.HCM. Ông cũng là một nhà bình duyệt bài báo tích cực cho tạp chí Computer & Security (ISSN: 0167-4048).

Hướng nghiên cứu hiện nay: Bảo mật di động, bảo mật mạng, học máy và học sâu.