MalDEX: An Explainable Malware Detection System Based on Ensemble Learning

Trong-Nghia To *†, Hien Do Hoang *†, Phan The Duy *†, and Van-Hau Pham*†

*Information Security Laboratory, University of Information Technology, Ho Chi Minh City, Vietnam
†Vietnam National University, Ho Chi Minh city, Vietnam
{nghiatt, hiendh, duypt, haupv}@uit.edu.vn

2023 International Conference on Multimedia Analysis and Pattern Recognition (MAPR) | 979-8-3503-2741-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/MAPR59823.2023.10288922

***Abstract*—Ensuring the security of information systems in the modern technology era is a critical requirement due to the increasing complexity and volume of malicious codes and threats. However, traditional signature-based approaches for malware detection suffer from limitations in identifying unknown malware, susceptibility to adversarial samples, and suboptimal performance. To overcome these challenges, leveraging Artificial Intelligence (AI) and Machine Learning (ML) in malware detection has emerged as a promising avenue for enhancing the accuracy and effectiveness of defense systems. This study introduces MalDEX, a novel Explainable black-box malware detection system. MalDEX leverages attribute extraction from Windows Portable Executable (PE) files and employs a range of classifiers, including Single Algorithms, Homogeneous Ensemble Algorithms, and Heterogeneous Ensemble Algorithms, to determine the presence of malware. To evaluate the models' quality and address interpretability concerns, we adopt the Kernel SHAP method. This method facilitates a comprehensive interpretation of the ML model's decision-making process by examining the input's contribution to decoding repetitive expectations. The experimental results demonstrate that the proposed system exhibits robust classification capabilities while providing descriptive visualizations and explanations elucidating the impact of features on prediction outcomes.**

***Index Terms*—Malware Detection, Ensemble Learning, Explainable AI, SHapley Additive exPlanations**

I. Introduction

In the modern technology era, ensuring the security of information systems has become an urgent requirement. The increasing number and complexity of malicious codes and threats pose significant challenges for security systems. To address these challenges, numerous methods and studies have been proposed and developed. For instance, Alieyan et al. [1] introduced a DNS rule-based approach for detecting Botnet, focusing on abnormal DNS query and response behaviors. Their approach utilized DNS query and response rules to accurately identify botnet activities. Similarly, Angelo et al. [2] proposed a recurring subsequences alignment-based algorithm that leverages associative rules to infer malware behaviors. By considering the probabilities of transitioning between API invocations and their timeline, the proposed approach extracts subsequences of API calls representative of common malicious behaviors. However, these signature-based approaches have become outdated and fail to meet current requirements. They suffer from limitations such as difficulty in detecting unknown malware, vulnerability to adversarial samples, and unsatisfactory performance.

In response to these limitations, the application of AI/ML to malware detection holds great promise for improving the performance and accuracy of defense systems. AI/ML methods can learn patterns, make automated predictions, and classify new threats based on learned knowledge.

Based on the Anomaly-based approach, in [3], the authors proposed a new approach where the system leverages significant features of activity to model normal and malicious behavior of users in Cloud-based settings. The experimental results demonstrate the potential of this method in security monitoring and detection of malicious activities. Another study by Stiborek et al. [4] proposed a model for analyzing malware behavior by examining its interactions with the operating system and network resources. To mitigate the impact of randomization techniques frequently used by malware authors to evade detection, the model employs an efficient clustering of resource names. Furthermore, it projects malware samples into a low-dimensional space that is suitable for classification using algorithms like Random Forest.

Regarding ML approaches, Bendiab et al. [5] proposed an innovative approach for analyzing IoT malware traffic. Their method utilizes Deep Learning (DL) and visual representation techniques to enable faster detection and classification of new malware, including zero-day malware. By operating at the package level, the detection of malicious network traffic in their approach significantly reduces the detection time. Similarly, Baptista et al. [6] presented a novel approach for malware detection that relied on binary visualization and self-organizing incremental neural networks. The researchers investigated the performance of this method in detecting malicious payloads across various file types. The experimental results revealed detection accuracies of 91.7% and 94.1% for ransomware in .pdf and .doc files, respectively. These studies have achieved great success in classifying and detecting risks and malware, however, they lack interpretation and the subsequent lack of trust in model outcomes.

To enhance the interpretability of DL models, in the study [7], Caforio et al. used the combination of the Grad-CAM explanations with the nearest-neighbor search to clarify normal and attack behavior in the network traffic and improved the accuracy of the CNN decisions. And in another work [8],

Mahbooba and his colleagues explored the interpretable side of the classification algorithm implemented for IDS using a Decision Tree that was considered a highly interpretable model. In the study conducted by Alenezi et al. [9], the researchers analyzed two cyber datasets consisting of five-class data. The authors employed three SHAP methods to explain the contributions of each feature in the models. The results confirmed that utilizing these classifiers to generate models proved to be a viable approach for detecting cybersecurity threats. However, these interpretive models lack practicality when using pre-processed datasets with few features and are only suitable for certain models. With such a small number of features, can the ML model provide a reasonable explanation for its decision and reuse that data in future studies?

Based on the previous studies, we have developed a novel Explainable black-box malware detection system named MalDEX. MalDEX extracts attributes from Portable Executable (PE) files in Windows and feeds them into a classifier. To enhance result diversity, we have designed classifiers based on three main types: Single Algorithm (SA), Homogeneous Ensemble Algorithm (HoEA), and Heterogeneous Ensemble Algorithm (HeEA). The classifier then determines whether the given PE file sample is malware or not. Furthermore, it is crucial to assess the quality of the models and identify any interpretability limitations. This analysis is achieved by examining the input's contribution to decoding repetitive expectations. To fully interpret the ML model's decision outputs for input PE files, we employ the SHApley Additive exPlanations (SHAP) KernelSHAP method. This method provides a comprehensive interpretation of the model's decisions.

In **Section II**, we provide an overview of our MalDEX system. **Section III** outlines the Dataset and Experimental configuration. The test results are presented in **Section IV**. Finally, **Section V** concludes the paper.

II. Methodology

A. Overview of Architecture

Fig. 1 presents an overview of the MalDEX architecture. It processes PE files as input and extracts their features, resulting in 2350-dimensional vectors. These features are then used by various classifiers in MalDEX, including SA-based classifiers (Decision Tree, Logistic Regression, K-Nearest Neighbors), Homogeneous Ensemble algorithm-based classifiers (Random Forest, Bagging, AdaBoost), and Heterogeneous Ensemble algorithm-based classifiers (Voting, Stacking). The proposed method provides classification results and explanations, allowing MalDEX to determine whether the input PE sample is malicious or benign. Additionally, SHAP with the violin summary plot is employed to elucidate the impact of features on the model's decision.

1) Feature Extracting module: The number of extracted features from PE files significantly impacts model processing. Insufficient features may not capture all the necessary characteristics for accurate predictions. Conversely, an excessive number of features can affect processing speed and calculations, such as MalDEX's SHAP values. In our study, we utilized a fixed number of 2350 dimensions for the extracted features. To handle infinite data types like imported functions, exported functions, and section information, we employed the FeatureHasher function to transform them into fixed-size vectors. Further information regarding the extracted profiles' names and dimensions is available in **Table I**.

TABLE I: Description of the extracted features

Feature name	Dimensions	Description
Byte Histogram	257	Byte histogram of the PE file
Byte-Entropy Histogram	256	Byte-entropy histogram of the PE file
String Information	103	Information about the used strings
Header Information	62	Information about COFF and the Optional Header
General File Information	9	General information about the PE file
Imported Functions	1280	Information about imported functions and libraries (using FeatureHasher)
Section Information	255	Properties of the used sections (using FeatureHasher)
Exported Functions	128	Information about exported functions (using FeatureHasher)

2) Ensemble-based Classifiers: Ensemble Learning, also known as committee-based learning, aims to combine multiple single algorithms (base models) to create an optimal predictive model. It leverages the strengths of different models and addresses weaknesses or inconsistencies in each model. By effectively integrating these models, Ensemble Learning can significantly enhance performance compared to using a single model. Ensemble Learning can be classified as Homogeneous and Heterogeneous Ensemble algorithms depending on the combination of learning algorithms. Homogeneous Ensembles employ the same type of learning algorithms, whereas Heterogeneous Ensembles leverage different learning algorithms.

In this study, we constructed nine classifiers categorized into three main groups: Single Algorithms (Decision Tree, Logistic Regression, K-Nearest Neighbors), Homogeneous Ensemble Algorithms (Random Forest, Bagging, AdaBoost), and Heterogeneous Ensemble Algorithms (Voting, Stacking).

The performance of SA-based classifiers influences the selection of estimators for Ensemble Algorithm-based classifiers. For Heterogeneous Ensemble algorithms, we employ three base estimators and choose the best-performing algorithm from the Single Algorithms and the Homogeneous Ensemble algorithms as the final estimator. Specifically, we utilize three models using Heterogeneous Ensemble algorithms: Voting(3, SA), Stacking(3, SA), and Stacking(3, HoEA). Voting(3, SA) employs the Voting algorithm with three Single Algorithms as estimators. Stacking(3, SA) utilizes the Stacking algorithm with three Single Algorithms as base estimators and the best Single Algorithm as the final estimator. Stacking(3, HoEA) employs the Stacking algorithm with three Homogeneous Ensemble Algorithms as base estimators and the best Homo-

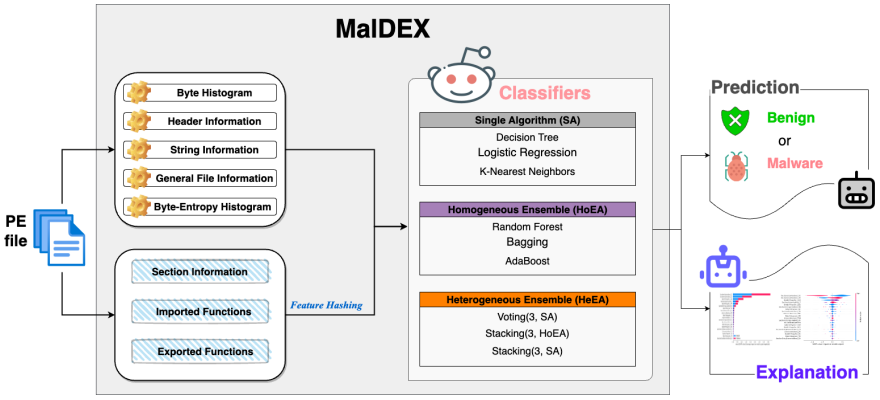

Fig. 1: The overview of MalDEX

geneous Ensemble Algorithms as the final estimator.

3) Prediction and Explanation: Besides predicting whether a sample is benign or malware, as determined by the aforementioned classifiers, MalDEX also offers explanations for these prediction outcomes. The purpose of these explanations is to enhance users' understanding of the significance of features in the prediction process. To generate the explanations, MalDEX utilizes Kernel SHAP to calculate the average marginal contribution of features to the model's prediction by considering all possible feature combinations.

KernelSHAP is a combination of Linear LIME and Shapley value taxonomies [10] that deliver highly accurate explanations and fits any model. This versatility is the reason why we utilize Kernel SHAP, as it allows us to apply a consistent and standardized comparison platform across all of our classifiers.

It employs a weighted linear regression technique to calculate feature importance, deriving Shapley values from game theory and coefficients from a local linear regression. By utilizing a weighted linear regression model as the local surrogate and employing a suitable weighting kernel, the regression coefficients of the LIME surrogate model estimate the SHAP values.

In order to apply the LIME approach, the loss function L is modified to update the explanation function g, as shown in **Eq.**(1):

$$L(f, g, \pi_x) = \sum_{z' \in Z} [f(h(z')) - g(z')]^2 \pi_x(z') \tag{1}$$

Besides, the Shapley kernel $pi_x(z')$ that recovers SHAP values is presented as follows:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)} \tag{2}$$

Where M is the number of features and $|z'|$ is the number of non-zero features in the simplified input z'.

We gets all the required Shapley values from the coefficients of g after fitting the model g by optimizing the loss function L.

III. Dataset and Experimental Configuration

A. Dataset

The dataset consists of 100,000 PE programs on the Windows platform. This collection includes 50,000 benign files obtained from a Windows 7 virtual machine (VM) and 50,000 malware files sourced from VirusTotal, encompassing various categories such as Adware, Trojans, Viruses, Ransomware, and Backdoors.

For training and evaluating classifiers in MalDEX, the dataset was divided into a train set and a test set using an 8:2 ratio. Specifically, 80,000 PE files were allocated for training, with 40,000 benign files and 40,000 malware files. The remaining 20,000 PE files were reserved for testing, comprising 10,000 benign files and 10,000 malware files.

B. Experimental Configuration

The system is built on a virtual machine running Ubuntu 18.04 LTS operating system, utilizing the Colab platform for efficient machine learning and data analysis tasks. The virtual machine's hardware configuration includes a 16-core Intel Xeon E5-2660 CPU with a clock speed of 2.0 GHz, 30 GB of RAM, and a 300GB hard drive. We leverage the high-RAM runtime option in Colab to optimize memory utilization for handling large datasets and complex computations.

For feature extraction, we utilize the Ubuntu virtual machine to ensure the safe handling of malicious code samples. On the other hand, training and model testing are performed on the Colab platform, utilizing its powerful computational capabilities. The system's source code was programmed in Python, utilizing main libraries such as PyTorch [11], Scikit-Learn [12], LIEF [13], SHAP [10], and several other supporting libraries.

After training our model, we utilize SHAP's KernelExplainer which is a model-agnostic method to generate SHAP values for a subset of 100 samples from the test dataset. Computing SHAP values for a large number of samples can be resource-intensive, hence the limited sample size. The computation time for generating SHAP values varies, ranging from 3 hours to nearly 6 days, particularly when using classifiers such as KNN or ensemble algorithms. This is due to the large

number of input features (2350 features) and the complexity of the model's parameters.

C. Metrics

1) Classification metrics: Classifier performance in malware recognition is assessed using metrics like AUC, accuracy, precision, recall, and F1-score. These metrics rely on the Confusion Matrix attributes: TP, TN, FP, and FN. Accuracy measures correct predictions, precision quantifies accurate malware detection, recall captures TP proportion, and F1-score combines precision and recall. AUC evaluates the TP rate and FP rate tradeoff. Higher metric values indicate a more effective malware detector. The formulas for the calculations are presented below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

2) Interpretability metrics: The Explanation results provided by MalDEX offer summary explanations for the test samples. To assess the interpretability of these explanations, we employed a human-based evaluation method [14] in our experiment to evaluate the generated explanations.

IV. Experimental Results

A. Classifier evaluation results

1) Single Algorithm-based Classifiers: **Table II** presents the performance of SA-based Classifiers. Among the three evaluated models, the Decision Tree (DT) model demonstrates superior performance. It exhibits high AUC, Accuracy, Precision, Recall, and F1-score, all-surpassing 96.7%, indicating its excellent ability to accurately classify and predict. Conversely, the models employing Logistic Regression (LR) and K-Nearest Neighbors (KNN) algorithms display average and good performance, respectively. The KNN-based model achieves an AUC of 81.44%, while the LR-based model attains an AUC of 66.74%. These results suggest a comparatively weaker class classification capability.

Based on the aforementioned evaluation outcomes, we employ the DT algorithm as estimators and the final estimator for the HoEA-based Classifiers and Stacking(3, SA) models, respectively.

TABLE II: Performance on Single Algorithm-based Classifiers

Model	AUC	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.6674	0.7566	0.8965	0.5802	0.7045
Decision Tree	**0.9720**	**0.9720**	**0.9670**	**0.9772**	**0.9721**
K-Nearest Neighbors	0.8144	0.7671	0.7132	0.8933	0.7932

2) Homogeneous Ensemble Algorithm-based Classifiers: The performance of the HoEA-based Classifiers is presented in **Table III**. The results indicate that all three models, namely Random Forest (RF), Bagging, and AdaBoost, exhibit good performance with all statistics exceeding 95.7%. Among the models, the RF algorithm demonstrates the highest values for AUC, Accuracy, Precision, Recall, and F1-score, all-surpassing 98.9%. Therefore, the RF algorithm is selected as the final estimator for the Stacking (3, HoEA) model.

TABLE III: Performance on Homogeneous Ensemble Algorithm-based Classifiers

Model	AUC	Accuracy	Precision	Recall	F1-score
Random Forest	**0.9989**	**0.9921**	**0.9890**	**0.9952**	**0.9921**
Bagging	0.9936	0.9787	0.9735	0.9842	0.9788
AdaBoost	0.9952	0.9733	0.9572	0.9909	0.9738

3) Heterogeneous Ensemble Algorithm-based Classifiers: The performance of the Heterogeneous Ensemble Algorithm-based Classifiers is presented in **Table IV**. The results indicate that the Voting, Stacking(3, SA), and Stacking(3, HoEA) models all exhibit good performance in malware classification and benignity, with indexes ranging from 91% to 99%. Among the models, the Stacking(3, HoEA) model achieves the highest index, with all indexes exceeding 97.7%.

TABLE IV: Performance on Heterogeneous Ensemble Algorithm-based Classifiers

Model	AUC	Accuracy	Precision	Recall	F1-score
Voting	0.9727	0.9317	0.9506	0.9107	0.9302
Stacking(3, SA)	0.9366	0.9356	0.9140	0.9616	0.9372
Stacking(3, HoEA)	**0.9971**	**0.9876**	**0.9779**	**0.9978**	**0.9877**

Summary: The classifiers utilizing ensemble algorithms demonstrate superior performance compared to the classifiers based on a single algorithm. All of these models exhibit strong classification and prediction capabilities, as indicated by AUC, Accuracy, Precision, Recall, and F1-score values surpassing 91%. Notably, certain models, such as RF and Stacking(3, HoEA), exhibit significantly superior performance, achieving a minimum of 97.7% for each evaluation metric.

B. Model Explanation

The explanations for each of the different classifiers are shown in **Fig. 2**, **Fig. 3**, **Fig. 4** (SA-based classifiers), **Fig. 5**, **Fig. 6**, **Fig. 7** (HoEA-based classifiers), **Fig. 8**, **Fig. 9**, and **Fig. 10** (HeEA-based classifiers). The **Section Information** characteristics have a significant impact on the model's ability to classify malicious patterns. This is evident from the fact that 8 out of 9 classifiers, except LR-based classifiers, utilize *"SectionVsizeHashed"* features, which represent the "actual" size of a section mapped to memory. These features, fixed in length by the Feature Hashing, consistently rank among the top 1 attributes influencing the classifier's decision. The RF model, in particular, has up to 8 *"SectionVsizeHashed"* characteristics among the top 20 most influential attributes affecting the prediction. These attributes gradually decrease

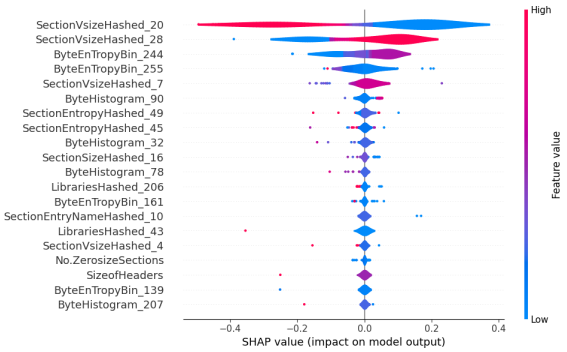

Fig. 2: Explanation of DT model

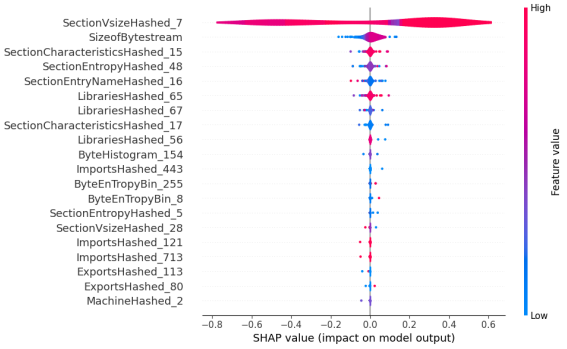

Fig. 3: Explanation of KNN model

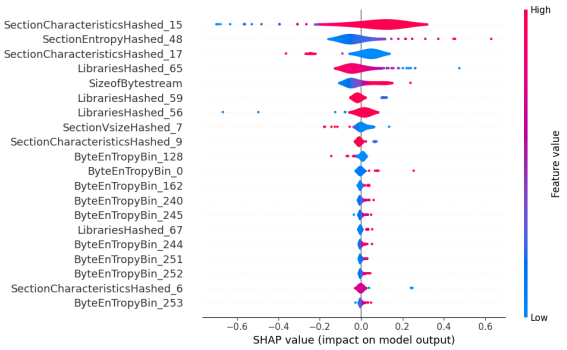

Fig. 4: Explanation of LR model

to 5 for models like Bagging and AdaBoost, 4 for models like DT, Stacking(3, HoEA), and Stacking(3, SA), and 3 for the Voting model. KNN and LR models, which exhibit the poorest performance, have only 2 and 1 *"SectionVsizeHashed"* attributes, respectively, among the top influencing attributes.

Furthermore, specific properties such as *"SectionEntropyHashed"* (associated with **Section Information**), *"ByteEntropy"* (related to **Byte-Entropy Histogram**), and *"LibrariesHashed"* (belonging to **Imported Functions**) also have an impact on the results, although not as significant as *"SectionVsizeHashed"*. Most models include these types of attributes in the summary plot of Kernel SHAP for classifying malicious and benign code. Specifically, all 9 models incorporate at least 1 attribute from the *"SectionEntropyHashed"* and *"ByteEntropy"* groups, while 8 out of 9 models include at least 1 attribute from the *"LibrariesHashed"* group. Additionally, it was observed that models using the HeEA algorithm value the *"ImportsHashed"* properties from the **Imported Functions** group, particularly when at least 3 attributes from this group are present.

To summarize, the best-performing models in each model type (SA, HoEA, HeEA) tend to prioritize properties related to Sections and Imports. This finding is positive and deserves recognition, especially considering that malicious or mutated samples often modify parts of the code that are not expected to affect the functionality of the malicious code, such as adding non-existent Imports or Sections. Furthermore, even when using Feature Hashing, MalDEX consistently generates

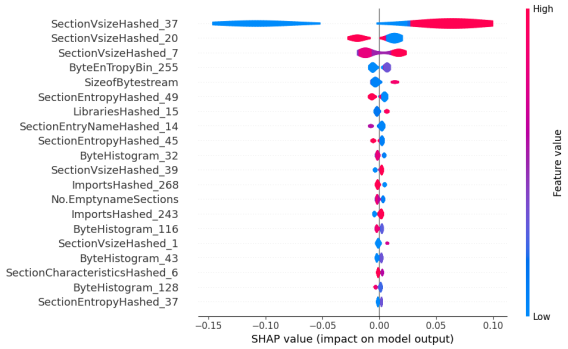

Fig. 5: Explanation of AdaBoost model

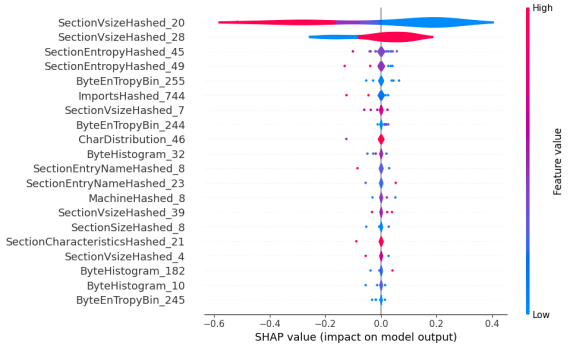

Fig. 6: Explanation of Bagging model

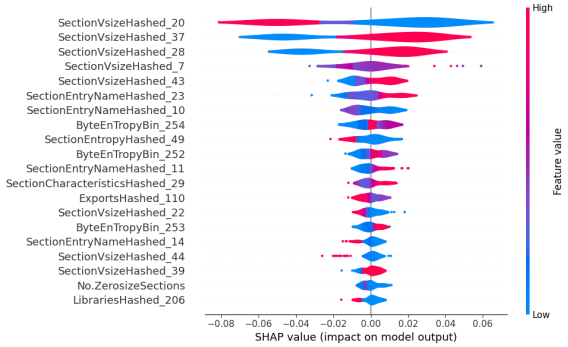

Fig. 7: Explanation of RF model

the explanations that capture the tendency to distribute values across the fixed-length feature vector. For example, the feature *"SectionVsizeHashed_7"* consistently ranks among the top 20 most influential features across all prediction results.

V. Conclusion

This paper introduces MalDEX, a novel Explainable black-box malware detection system that leverages attribute extraction from Windows Portable Executable files and employs various classifiers. By adopting the Kernel SHAP method, we achieve comprehensive model interpretation and evaluation. The experimental results demonstrate the system's robust classification capabilities and its ability to provide descriptive visualizations and explanations for prediction outcomes. In the future, we intend to develop a MalDEX-based mutation prototyping method to harness its power. Furthermore, expanding the dataset for training and testing purposes will be crucial in ensuring more precise and reliable interpretation and prediction outcomes.

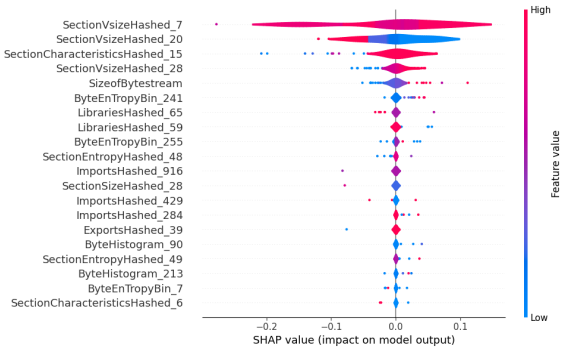

Fig. 8: Explanation of Voting model

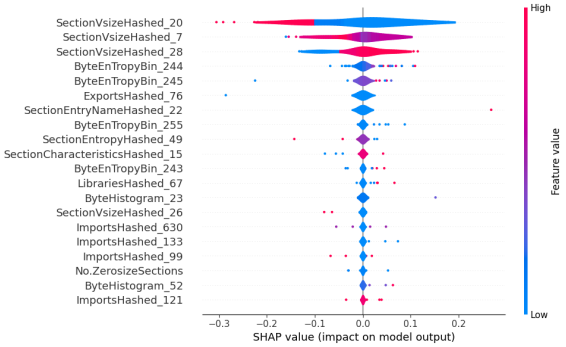

Fig. 9: Explanation of Stacking(3, SA) model

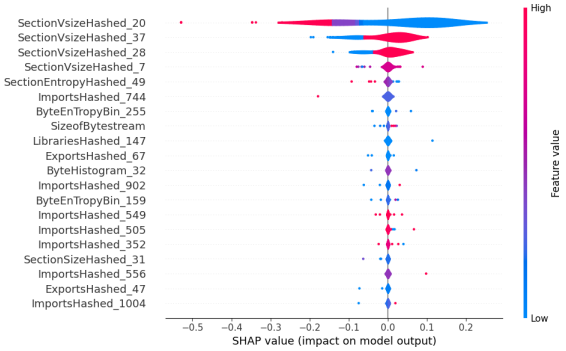

Fig. 10: Explanation of Stacking(3, HoEA) model

Acknowledgement

This research is funded by University of Information Technology - Vietnam National University HoChiMinh City under grant number of D1-2023-31.

References

[1] K. Alieyan, A. Almomani, M. Anbar, M. Alauthman, R. Abdullah, and B. B. Gupta, "Dns rule-based schema to botnet detection," *Enterprise Information Systems*, vol. 15, no. 4, pp. 545–564, 2021.

[2] G. D'Angelo, M. Ficco, and F. Palmieri, "Association rule-based malware classification using common subsequences of api calls," *Applied Soft Computing*, vol. 105, p. 107234, 2021.

[3] M. Rabbani, Y. L. Wang, R. Khoshkangini, H. Jelodar, R. Zhao, and P. Hu, "A hybrid machine learning approach for malicious behaviour detection and recognition in cloud computing," *Journal of Network and Computer Applications*, vol. 151, p. 102507, 2020.

[4] J. Stiborek, T. Pevnỳ, and M. Rehák, "Multiple instance learning for malware classification," *Expert Systems with Applications*, vol. 93, pp. 346–357, 2018.

[5] G. Bendiab, S. Shiaeles, A. Alruban, and N. Kolokotronis, "Iot malware network traffic classification using visual representation and deep learning," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 444–449.

[6] I. Baptista, S. Shiaeles, and N. Kolokotronis, "A novel malware detection system based on machine learning and binary visualization," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2019, pp. 1–6.

[7] F. P. Caforio, G. Andresini, G. Vessio, A. Appice, and D. Malerba, "Leveraging grad-cam to improve the accuracy of network intrusion detection systems," in *Discovery Science: 24th International Conference, DS 2021, Halifax, NS, Canada, October 11–13, 2021, Proceedings 24*. Springer, 2021, pp. 385–400.

[8] B. Mahbooba, M. Timilsina, R. Sahal, and M. Serrano, "Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model," *Complexity*, vol. 2021, pp. 1–11, 2021.

[9] R. Alenezi and S. A. Ludwig, "Explainability of cybersecurity threats data using shap," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 01–10.

[10] X. Man and E. P. Chan, "The best way to select features? comparing mda, lime, and shap," *The Journal of Financial Data Science*, vol. 3, no. 1, pp. 127–139, 2021.

[11] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Introduction to pytorch," *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pp. 27–91, 2021.

[12] E. Bisong and E. Bisong, "Introduction to scikit-learn," *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pp. 215–229, 2019.

[13] R. Thomas, "Lief-library to instrument executable formats," *Retrieved February*, vol. 22, p. 2022, 2017.

[14] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," in *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 2018, pp. 80–89.