Journal of Modern Technology and Engineering
Vol.9, No.3, 2024, pp.165-177
https://doi.org/10.62476/jmte93165

Jomard Publishing

ADVANCED MACHINE LEARNING AND INTERPRETABILITY FOR WINDOWS MALWARE DETECTION

Elshan Baghirov*

Institute of Information Technology, Baku, Azerbaijan

Abstract. Machine learning (ML) has emerged as a powerful tool for detecting and mitigating malware, addressing the evolving challenges in cybersecurity. This paper presents a comprehensive overview of ML techniques applied to Windows Portable Executable (PE) malware detection, spanning from theoretical foundations to practical implementations. Theoretical underpinnings such as feature engineering, model selection, and evaluation metrics are explored, followed by discussions on practical aspects including data preprocessing, model training, and deployment considerations. The experimental setup using the Dataiku platform is detailed, and seven ML models are evaluated on both binary and multiclass classification tasks before and after applying Principal Component Analysis (PCA). The performance and interpretability of these models are analyzed using SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations). By synthesizing insights from theory and practice, this paper aims to provide a comprehensive understanding of ML approaches for Windows PE malware detection and guide future advancements in cybersecurity.

Keywords: Windows Portable Executable, malware detection, machine learning, explainable AI.
AMS Subject Classification: 68M15.
Corresponding author: Elshan Baghirov, Institute of Information Technology, B. Vahabzade 9A, Baku, Azerbaijan, Tel.: +994514441933, e-mail: *elsenbagirov1995@gmail.com*
Received: 29 October 2024; Revised: 20 November 2024; Accepted: 5 December 2024; Published: 25 December 2024.

1 Introduction

Malware's impact on individuals, organizations, and society is significant and multifaceted. Cyberattacks, including inserting ads into websites and stealing confidential data, often involve malicious software such as viruses, worms, trojans, and ransomware (Adnaan et al., 2023; Baghirov, 2023; Mbunge et al., 2023). These are meant to disrupt, compromise, or gain unauthorized access to computer systems (Jamar et al., 2017), leading to substantial financial, operational, and reputational harm. For individuals, malware poses risks such as identity theft and financial loss, while organizations face disrupted operations (Youssef et al., 2022), data breaches, and potential legal consequences. Societal impacts include critical infrastructure disruption, compromised national security, and essential service disruption. As malware evolves, robust detection and prevention measures become increasingly crucial to safeguard the digital landscape from its wide-ranging effects. According to the 2024 report from Malwarebytes (2024), ransomware attacks increased by 68%, with malware constituting 11% of detections on Macs. The AV-TEST Institute reports over 450,000 new instances of malicious programs and potentially unwanted applications daily.

How to cite (APA): Baghirov, E. (2024). Advanced machine learning and interpretability for windows malware detection. *Journal of Modern Technology and Engineering*, *9*(3), 165-177 https://doi.org/10.62476/jmte93165

Malware, a persistent and ever-evolving cyber threat, has necessitated the adoption of various protective measures (Aboaoja et al., 2022). Cybersecurity experts employ diverse strategies to detect and mitigate its impact, including static and dynamic analysis methods. Static analysis involves examining code characteristics without execution, enabling quick scanning of large file volumes. However, it may struggle with polymorphic or obfuscated malware and yield false positives. Dynamic analysis entails executing malware in a controlled environment to observe its behavior in real-time, offering insights into its actual behavior but requiring significant computational resources and potentially introducing detection delays. Despite the strengths of static and dynamic analysis, malware authors continually evolve tactics to evade detection. Future research may focus on developing hybrid approaches, combining both methods and enhancing the interpretability of detection models through explainable AI. Success in this innovation would improve how detecting and combating malicious software are approached. The following are the main contributions of this work:

- *Evaluation of machine learning models on both binary and multiclass datasets, with and without PCA*
- *Insights into feature importance and model interpretability*
- *Practical considerations for real-world deployment*
- *Identification of future research directions*

Our paper is arranged as follows: Section 2 provides the theoretical foundation and fundamental concepts concerning malware. Section 3 conducts a review of relevant literature. Section 4 outlines the experimental setup and analysis. Lastly, we explain our results in Section 5 and conclude the paper in Section 6.

2 Background and basic concepts

2.1 Malware Obfuscation Methods

Malware obfuscation is like a cloak that cybercriminals use to hide their malicious software from antivirus scanners (Chen et al., 2021). It's a sneaky tactic that helps them slip past security defenses. Some examples of obfuscation techniques are encryption, compression, encoding, polymorphism, metamorphism, and code injection (You and Yim, 2010). Common malware obfuscation techniques has been shown in Table 1.

Table 1: Malware Obfuscation Methods

Method	Description
Encryption	The process of transforming data (plaintext) to make it unreadable except to those possessing a key (Li, 2009).
Compression	Reduces data size by encoding it in a more efficient representation, minimizing file sizes and evading detection (Huang et al., 2024).
Encoding	Transforms data into another format using a scheme such as Base64, obscuring content and making it harder to detect.
Polymorphism	Modifies its code each time it infects a new host while maintaining the same functionality, making it difficult for antivirus software to identify and detect (Selamat et al., 2016).
Oligomorphism	Shares similarities with polymorphic malware but has limited variations, producing a finite number of distinct forms (You and Yim, 2010).
Metamorphism	Changes its code and underlying structure, making it even more challenging to detect and analyze (Mirzazadeh et al., 2015).
Code Injection	Involves inserting malicious code into a legitimate program or process to execute unauthorized actions, bypassing security measures and gaining unauthorized access to systems (Erfina et al., 2023).

As outlined by Huidobro et al. (2017), deploying some of these obfuscation techniques has become increasingly prevalent, particularly in updating malware. Automated obfuscation methods have emerged as a preferred choice, offering cybercriminals a reliable and efficient means of perpetuating their illicit activities and making their malware updates harder to detect.

2.2 Windows PE File Structure

The term "Portable Executable" denotes the format's adaptability and versatility across various architectures, making it a foundational element in Windows-based systems (Microsoft Build, 2024). PE files consist of headers and sections that play vital roles in the execution and functionality of applications, libraries, and dynamically linked modules. The file structure begins with the DOS header, followed by the PE header, which contains essential metadata like the optional header and section table. The optional header provides information about the file's architecture and entry point, while the section table lists the characteristics and locations of each section. The sections include the code section, housing executable instructions, the data section for initialized data, the resource section for non-executable resources, and the import section for external functions and libraries. Understanding these components helps analysts detect and mitigate security threats. Structure of Windows PE file has been shown in Figure 1.

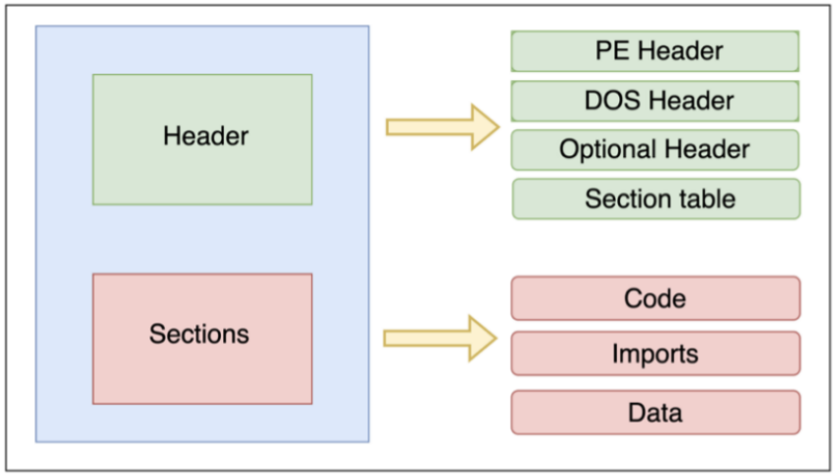

Figure 1: Windows PE file structure

3 Literature Review

Detecting and analyzing malicious code in PE files remains an essential and ongoing cybersecurity challenge. The continuous evolution of malware necessitates the development of more advanced and adaptive detection methods. Traditional signature-based techniques, while effective in some scenarios, struggle to keep up with the rapid generation of new malware variants. To address these limitations, researchers have increasingly turned to machine learning approaches, which offer the potential for more dynamic and robust malware detection capabilities.

Initial efforts in malware detection predominantly relied on signature-based techniques, where known malware signatures were used to identify threats. Although effective for known malware, these methods are inadequate against new or obfuscated malware. To enhance detection accuracy, researchers like Belaoued and Mazouzi (2016) explored Chi-Square analysis, which, while providing statistical insights, still falls short in dynamic and real-time malware detection scenarios.

Machine learning has emerged as a powerful tool to overcome the shortcomings of traditional methods. Various ML models have been proposed, each contributing unique advantages.

For instance, Azeez et al. (2021) advocated for the use of ensemble learning, which combines multiple learning models to improve accuracy and robustness. Their study utilized a dataset of 19,611 malicious samples and demonstrated that an ensemble of fully connected dense artificial neural networks (ANN) and one-dimensional convolutional neural networks (1-D CNN), with ExtraTrees as the final-stage classifier, achieved superior classification performance. This ensemble method outperformed both conventional and other advanced techniques, highlighting the potential of ensemble learning in malware detection.

Similarly, Catak et al. (2020) developed a classification method based on the behavior of malware, utilizing API calls made on the Windows operating system. Their dataset included various types of malicious software such as Adware, Backdoor, Downloader, Dropper, Spyware, Trojan, Virus, and Worm. Using Long Short-Term Memory (LSTM) networks, a widely used method for sequential data, they achieved an accuracy of up to 95% with a 0.83 F1-score. This study demonstrated the efficacy of LSTM models in capturing the sequential nature of API call data for malware classification.

In a novel approach, researchers Shaukat et al. (2024) employed visualization techniques to identify malware through Windows portable executables. This framework, by eliminating the need for traditional feature extraction, significantly reduced training costs for deep learning models while achieving an accuracy of 99.30%. Despite its efficiency and lightweight design, the framework's vulnerability to adversarial attacks was highlighted by subsequent studies Imran et al. (2024), which demonstrated how adversarial training could enhance the system's resilience against such threats.

The study suggested in Chen et al. (2021) addresses the challenge of obfuscation in Android malware detection by employing code deobfuscation techniques to retrieve concealed information, enhancing the detection system's effectiveness. The proposed model, incorporating obfuscation-invariant features and interaction terms, achieves a 99.55% accuracy and a 94.61% F1-score using the Drebin dataset. However, the use of an older dataset raises concerns about the model's robustness and applicability to current malware threats.

Despite the promising advancements in ML-based malware detection, significant challenges remain. One critical issue is the system's susceptibility to adversarial attacks, which can manipulate malware to evade detection by even the most sophisticated models. Additionally, the integration of ML models into real-world systems requires careful consideration of data preprocessing, model training, and deployment strategies to ensure optimal performance and security.

4 Experimental setup and analysis

4.1 Dataset description

The dataset comprises Windows PE samples with four distinct feature sets, each stored in a separate CSV file (Yousuf, 2023). These feature sets provide valuable insights into the behavior and characteristics of various malware families. The dataset is labeled, with each sample associated with one of seven malware types or families, as well as a benign category. The sample size for each malware type is shown in Table 2.

Table 2: Sample Size for Each Malware Type or Family

Malware Type/Family	Sample Size	Proportion (%)
Spyware	3699	11.9%
SnakeKeyLogger	4227	13.6%
BankingTrojan	5076	16.3%
RAT	4955	15.9%
Downloader	4643	14.9%
RedLineStealer	5022	16.1%
Benign	1877	6.0%

The dataset features and feature size are structured as follows:

- **DLLs Imported Feature Set**: This feature set contains information about the Dynamic Link Libraries (DLLs) imported by each malware sample. There are 629 features in this set.
- **API Functions Feature Set**: This feature set includes details about the Application Programming Interface (API) functions called by the malware samples. There are 792 features in this set.
- **PE Header Feature Set**: This feature set provides values for 52 fields extracted from each sample's PE header. The fields are labeled in the CSV file, offering insights into various aspects of the PE structure, such as file size, entry point address, and section characteristics. There are 52 features in this set.
- **PE Section Feature Set**: This feature set comprises values for nine fields extracted from ten different PE sections of each sample. Similar to the PE header feature set, the fields are labeled, providing information about the characteristics and attributes of each PE section, such as virtual size, raw size, and entropy. There are 89 features in this set.

  Overall, the combined feature set used for analysis includes 1562 features, encompassing various attributes that provide comprehensive insights into the characteristics and behaviors of the malware samples.

4.2 Dataset Preprocessing

The data preprocessing section plays a pivotal role in our machine learning model. We applied standard scaling to numerical columns, ensuring uniformity in their scales. This aids in the convergence of machine learning algorithms and significantly improves model performance. We also excluded columns with a high proportion of identical values from the dataset to mitigate potential bias and reduce computational overhead. Furthermore, we removed rows containing missing values to maintain data integrity and prevent imputation-induced inaccuracies. These preprocessing steps are not just important, they are crucial for enhancing the quality and suitability of the dataset for subsequent machine learning model training and evaluation.

4.3 Experimental Setup

Our analysis and modeling experiments were conducted within the powerful Dataiku platform. This integrated environment is renowned for its capabilities in advanced data analytics and machine learning tasks. With a suite of tools and functionalities tailored for data preparation, feature engineering, model building, and evaluation, Dataiku provided a robust ecosystem for our research. The version utilized for this study is **Dataiku 10.0.5 (licensed)**, with the notebook server running version 5.4.0-dku10.0-0 and Python 3.6.8.

4.4 Applied Models and Metrics for Malware Detection

In the pursuit of effective malware detection, a variety of machine learning models have been employed to analyze and classify malicious software. Notable models include Decision Trees, K-Nearest Neighbors (KNN), LightGBM, Logistic Regression, Random Forests, Support Vector Machines (SVM), and XGBoost.

The selection of these models for the interpretability analysis is grounded in their diverse approaches to handling data, each offering unique strengths and weaknesses that contribute to a comprehensive evaluation. Decision Trees are chosen for their inherent simplicity and ease of interpretability, making them a valuable baseline for understanding model decisions. KNN is included to represent instance-based learning methods, providing insights into how similarity

measures impact detection performance. LightGBM and XGBoost are selected due to their proven efficiency and accuracy in handling large datasets and complex feature interactions, making them ideal for high-performance comparisons. Logistic Regression offers a linear perspective, useful for understanding how well simple linear relationships can capture malware patterns. Random Forests, with their ensemble approach, demonstrate the power of combining multiple decision trees to enhance predictive accuracy and robustness. SVM are included for their effectiveness in high-dimensional spaces, highlighting the impact of margin maximization on classification tasks.

Given the large feature set in our dataset, we applied PCA for dimensionality reduction to enhance computational efficiency and model performance. Consequently, we evaluated the performance of these models both with and without the application of PCA. We also tested the models on both binary and multiclass classification tasks to comprehensively assess their effectiveness across different types of classification problems.

To evaluate the performance of these models, several key metrics are utilized: accuracy measures the proportion of correctly classified instances; precision assesses the accuracy of the positive predictions; recall indicates the ability to identify true positive instances; F1 score provides a balance between precision and recall; and the Receiver Operating Characteristic Area Under the Curve (ROC AUC) measures the overall ability of the model to distinguish between classes; and training time, which evaluates the computational efficiency of each model.

4.5 Explainable AI Methods

Integrating explainability into our approach, we aimed to make our machine learning models for detecting obfuscated malware more transparent and interpretable. To achieve this, we utilized SHAP (Lundberg and Lee, 2017) and LIME (Ribeiro et al., 2016). These techniques enabled us to comprehend and interpret the decisions made by sophisticated models, fostering transparency and trust.

Derived from cooperative game theory, SHAP values offer a unified measure of feature importance. They illustrate how each feature contributes to the prediction by averaging over all possible feature combinations. SHAP maintains three core properties: local accuracy, missingness, and consistency. Local accuracy ensures that the sum of feature attributions equals the model output for each instance. Missingness guarantees that features absent from the model do not affect the output. Consistency ensures that if a feature's contribution increases or remains the same when the model changes, its attribution does not decrease. Various methods, such as Kernel SHAP, can be used to approximate SHAP values for different model types (Lundberg and Lee, 2017).

The SHAP value formula is given in Equation 1:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[f(S \cup \{i\}) - f(S) \right] \tag{1}$$

In this formula, ϕ_i represents the SHAP value for feature i, S is a subset of all features N excluding i, $f(S \cup \{i\})$ denotes the prediction including feature i in subset S, and $f(S)$ is the prediction without feature i. The term $\frac{|S|!(|N|-|S|-1)!}{|N|!}$ is a weighting factor based on the subset size.

LIME interprets classifier predictions by locally approximating them with a simpler model. It perturbs the data around the instance to be explained and fits a straightforward, interpretable model (such as linear regression) on these perturbed samples. This local model offers insights into how each feature affects the prediction in the vicinity of the instance. LIME balances the need for interpretability with fidelity to the original model (Ribeiro et al., 2016).

The formula for the LIME explanation model is presented in Equation 2:

$$\xi(x) = \arg\min_{g \in G} \sum_{z \in Z} \pi_x(z) \left(f(z) - g(z)\right)^2 + \Omega(g) \tag{2}$$

In this equation, $\xi(x)$ is the explanation model for the instance x, $g \in G$ represents a family of interpretable models, $\pi_x(z)$ is a proximity measure between z and x, $f(z)$ is the prediction of the complex model, and $g(z)$ is the prediction of the interpretable model. The term $\Omega(g)$ serves as a regularization term to ensure the simplicity of the explanation model.

5 Results

5.1 Performance Assessment of Models with and without PCA

After assessing the performance of machine learning models trained with default parameters and utilizing all available features, LightGBM emerged as the top performer among the models evaluated, demonstrating superior performance in malware detection tasks. We also explored the application of PCA to reduce the dimensionality of our feature space while preserving as much information as possible. Specifically, we limited the feature set to 25 principal components and evaluated the impact on the performance of our machine-learning models for malware detection.

Comparing the results of models before and after applying PCA with 25 components are presented in Table 3 and Table 4. We evaluated the models on both binary and multiclass datasets. While LightGBM and Random Forest maintain their positions as top performers in both scenarios, there are noticeable efficiency improvements. Specifically, the training time for these models has decreased compared to the previous scenario. Decision Tree and Logistic Regression models exhibit slight decreases in accuracy after PCA, emphasizing the trade-off between dimensionality reduction and model performance. Notably, SVM's accuracy sees a considerable drop, highlighting the sensitivity of specific models to feature reduction. Overall, PCA enhances efficiency without significant compromise in performance for most models, with LightGBM and Random Forest continuing to excel in malware detection tasks.

Table 3: Multiclass Models Performance with All Features and After Applying PCA

Model	Time	Accuracy	Precision	Recall	F1 Score	ROC AUC
All Features						
Decision Tree	1m 48s	0.72	0.74	0.72	0.72	0.94
KNN (k=5)	1m 2s	0.84	0.84	0.84	0.84	0.96
LightGBM	**2m 33s**	**0.90**	**0.90**	**0.90**	**0.90**	**0.99**
Logistic Regression	4m 35s	0.74	0.74	0.72	0.72	0.95
Random Forest	2m 58s	0.89	0.89	0.89	0.89	0.99
SVM	33m 59s	0.72	0.76	0.70	0.68	0.95
XGBoost	3m 38s	0.85	0.86	0.86	0.86	0.98
After Applying PCA						
Decision Tree	14s	0.66	0.68	0.65	0.65	0.91
KNN (k=5)	41s	0.83	0.83	0.83	0.83	0.95
LightGBM	34s	0.86	0.86	0.85	0.86	0.98
Logistic Regression	52s	0.59	0.63	0.56	0.54	0.90
Random Forest	**48s**	**0.86**	**0.86**	**0.86**	**0.86**	**0.98**
SVM	3m 9s	0.56	0.60	0.53	0.52	0.90
XGBoost	32s	0.81	0.82	0.81	0.81	0.97

5.2 Interpretability analysis

In this section, we focus on interpreting the predictions of the LightGBM model, a booster algorithm known for its complex design and high predictive performance in malware detection

Table 4: Binary Models Performance with All Features and After Applying PCA

Model	Time	Accuracy	Precision	Recall	F1 Score	ROC AUC
All Features						
Decision Tree	1m 5s	0.96	0.63	0.92	0.75	0.97
KNN (k=5)	15m 4s	0.99	0.94	0.85	0.89	0.96
LightGBM	**1m 19s**	**1.00**	**0.97**	**0.96**	**0.96**	**1.00**
Logistic Regression	7m 48s	0.97	0.79	0.72	0.76	0.97
Random Forest	1m 20s	0.99	0.94	0.96	0.95	**1.00**
XGBoost	1m 20s	0.99	0.94	0.91	0.93	**1.00**
SVM	25m 56s	0.95	0.65	0.58	0.61	0.94
After Applying PCA						
Decision Tree	15s	0.97	0.73	0.78	0.75	0.96
KNN (k=5)	47s	0.98	0.95	0.80	0.87	0.97
LightGBM	**23s**	**0.99**	**0.95**	**0.90**	**0.93**	**0.99**
Logistic Regression	2m 19s	0.96	0.65	0.86	0.74	0.94
Random Forest	1m 35s	0.99	0.94	0.90	0.92	0.99
SVM	9m 38s	0.96	0.70	0.59	0.64	0.94
XGBoost	29s	0.99	0.92	0.87	0.90	0.99

tasks. We start by analyzing the importance of features to understand which features significantly influence the model's predictions. Subsequently, we employ SHAP and LIME to gain deeper insights into the decision-making process of the LightGBM model.

Feature importance provides a quantitative measure of the relative importance of each feature in making predictions. The LightGBM model inherently provides feature importance scores based on how useful each feature is in reducing impurity across all trees in the ensemble. Features like time datestamp, characteristics, checksum, etc., are identified by the LightGBM model, which is crucial for distinguishing between malware and benign software. The top 20 features have been shown in Figure 2.

Figure 3 shows the SHAP summary plot for the spyware class.

The SHAP force plot in Figure 4 explains how each feature contributes to the LightGBM model's prediction for a specific instance in a multiclass classification scenario. The plot visualizes how various features influence the model's output, pushing it towards different classes, with higher values indicating a stronger tendency towards a particular class.

The SHAP force plot depicted in Figure 4 provides a detailed visualization of how each feature contributes to the model's prediction for a particular instance. The base value, shown at the center of the plot, represents the average model output over the training dataset, essentially reflecting what the model would predict if no features were present. In this instance, the base value is set at 0, and the model's final prediction is -6.60, indicated at the central point labeled as "f(x) = -6.60."

Features contributing to increasing the model's prediction are represented by the red arrows on the left, pushing the prediction higher from the base value, while features contributing to decreasing the prediction are represented by the blue arrows on the right, pulling the prediction lower. The length of each arrow reflects the magnitude of the feature's impact. In this plot, "SizeOfImage" and "TimeDateStamp" are the features pushing the prediction higher with contributions of 0.27 and 0.64, respectively, while "msvbvm60.dll" is a feature pulling the prediction lower with a contribution of -2.91.

This visualization helps in understanding the relative importance and effect of each feature in the prediction process, thereby enhancing model interpretability. By illustrating how specific features influence the output, SHAP summary plots provide valuable insights into the decision-making process of complex machine learning models, ensuring transparency and fostering trust in the model's predictions.

The image presented in Figure 5 shows the results of a malware detection model that classifies a sample into various malware categories, including Spyware, SnakeKeyLogger, RAT (Re-

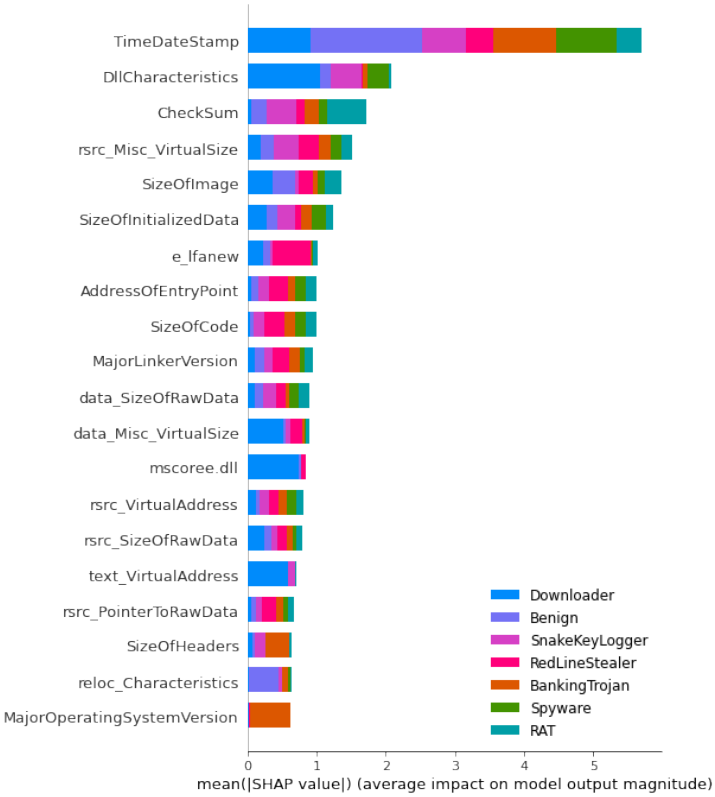

Figure 2: Top 20 features of LightGBM model

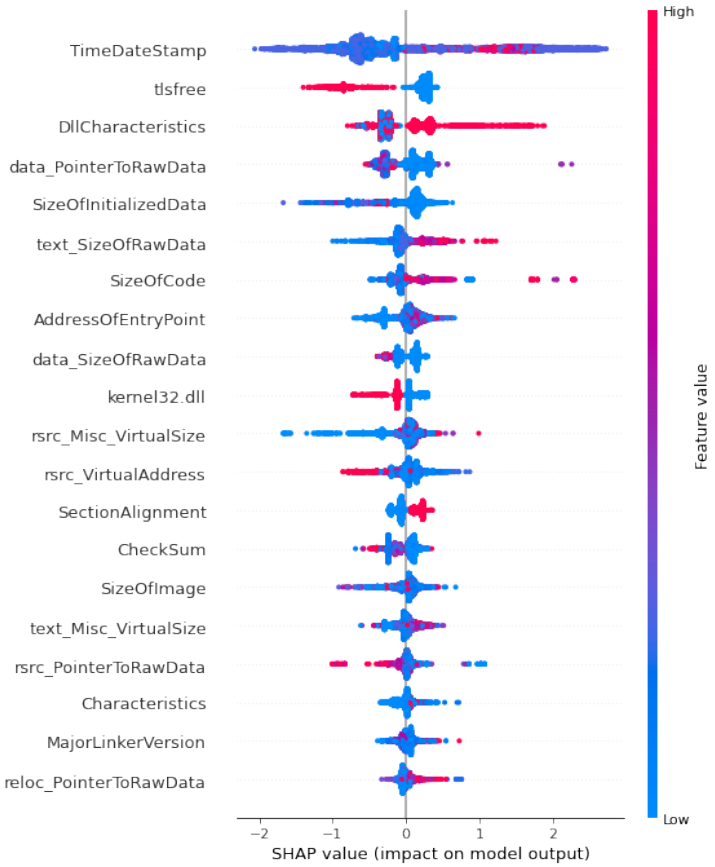

Figure 3: SHAP summary plot for spyware class.

Table 5: Comparison of Other Malware Detection Approaches

Author	Dataset	Type	Extracted features	Result	Strengths and Limitations
Shukla et al. (2019)	19 612 samples (14 599 malware) with 75 features	binary	PE header	95% accuracy	Unable to predict malware family, not robust to obfuscation, lack of robustness
Aurangzeb & Aleem (2023)	142 128 samples (70 127 malware) with 289 features	binary	System calls, static features	93% on emulator 94% on real without obfuscation 91% on emulator as well as on real with obfuscation	Dataset is old, unable to predict malware family, robust to obfuscation
Islam et al. (2020)	123 453 samples (5 560 malware)	binary	opcodes	97% accuracy	Dataset is old, unable to predict malware family, not robust to obfuscation, lack of robustness
Our work	29 499 samples (7 classes) with 1 562 features	binary, multiclass	DLL, API, PE header, PE sections	binary - 96% f1-score multiclass - 90% f1-score	binary and malware family prediction, resilient to obfuscation methods

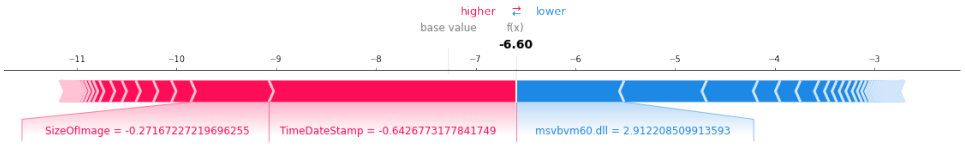

Figure 4: SHAP force plot.

mote Access Trojan), BankingTrojan, and others. The model predicts with high confidence

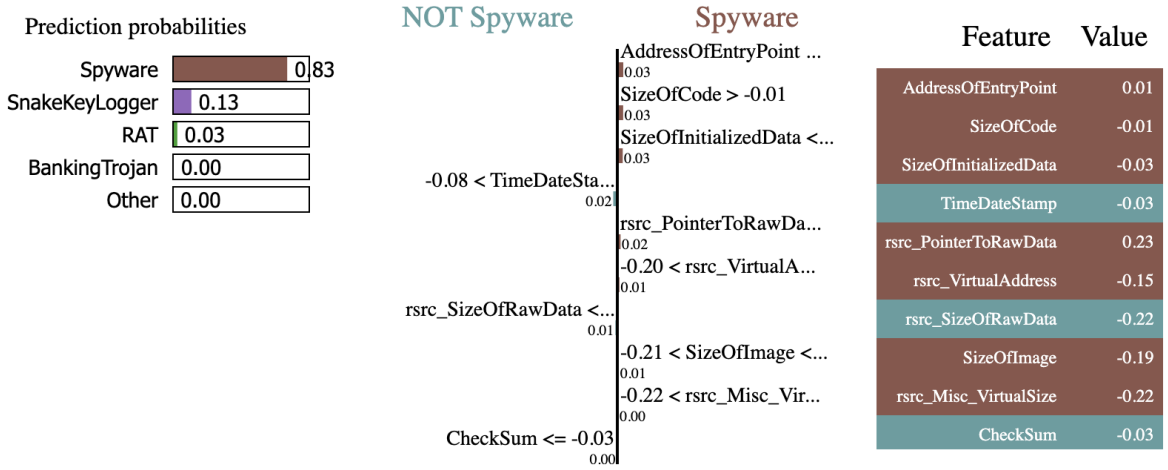

Figure 5: Interpreting Model Predictions with LIME

(83%) that the sample is Spyware. The probabilities for other categories are much lower, with SnakeKeyLogger at 13%, RAT at 3%, and both BankingTrojan and Other at 0%.

The bar chart in the middle indicates the contribution of different features to the model's

decision, highlighting the impact of each feature on predicting whether the sample is Spyware or NOT Spyware. Features contributing to the Spyware classification include AddressOfEntryPoint, SizeOfCode greater than -0.01, SizeOfInitializedData less than a certain threshold, rsrc_PointerToRawData, rsrc_VirtualAddress less than -0.20, SizeOfImage less than -0.21, and rsrc_Misc_VirtualSize less than -0.22.

On the other hand, features indicating the sample is NOT Spyware include TimeDateStamp less than -0.08, rsrc_SizeOfRawData less than -0.21, and CheckSum less than or equal to -0.03. The table on the right lists the actual values of these features, which are: AddressOfEntryPoint at 0.01, SizeOfCode at -0.01, SizeOfInitializedData at -0.03, TimeDateStamp at -0.03, rsrc_PointerToRawData at 0.23, rsrc_VirtualAddress at -0.15, rsrc_SizeOfRawData at -0.22, SizeOfImage at -0.19, rsrc_Misc_VirtualSize at -0.22, and CheckSum at -0.03.

The comparison of different malware detection approaches is presented in Table 5. This table provides a concise overview of various datasets, classification types, feature extraction methods, results, and the strengths and limitations of each approach, including our work. Although our detection rate is slightly lower, our method is resilient to obfuscation techniques and capable of predicting malware families, demonstrating its robustness and practicality for real-world applications.

6 Conclusions and Future works

In this paper, we have explored machine learning techniques for malware detection, bridging the gap between feature engineering, model selection, and evaluation metrics. Our study highlights their critical roles in enhancing the accuracy and effectiveness of malware detection systems. We also examined practical aspects such as data preprocessing, model training, and deployment considerations, highlighting the challenges faced in real-world scenarios.

Our experimental results demonstrate that the LightGBM model consistently outperforms other algorithms in terms of accuracy, precision, recall, and F1 score. We evaluated the models on both binary and multiclass datasets and applied PCA to reduce the dimensionality of our feature space while preserving as much information as possible. Specifically, we limited the feature set to 25 principal components and assessed the impact on the performance of our machine learning models for malware detection. Comparing the results before and after applying PCA revealed noticeable efficiency improvements, particularly in training time, without significant compromise in performance for most models.

Feature importance analysis and interpretability techniques like SHAP and LIME provide valuable insights into the model's decision-making process, making it easier to understand and trust its predictions.

Addressing the challenges of deploying ML models in real-world environments is critical and involves dealing with issues like data drift and adversarial attacks and maintaining model performance over time. Another important direction is developing more scalable and efficient models that can handle large volumes of data in real-time without compromising accuracy. This suggested direction includes optimizing algorithms for faster training and inference times. Enhancing the diversity and quality of datasets used for training and evaluation is crucial. Incorporating more varied malware samples and benign applications from different environments will improve the generalizability of detection models.

References

Aboaoja, F.A., Zainal, A., Ghaleb, F.A., Al-Rimy, B.A.S., Eisa, T.A.E., & Elnour, A.A.H. (2022). Malware detection issues, challenges, and future directions: A survey. *Applied Sciences*, *12*(17), 8482. https://doi.org/10.3390/app12178482

Adnaan, S., Tejasri, G., Gowtham, M., Reddy, S., Venkata, A., & Krishna, P. (2023). A detailed study on preventing the malicious URLs from cyber attacks. *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 414–421.

Aurangzeb, S., Aleem, M. (2023). Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism. *Scientific Reports*, **13**(1), 3093. https://doi.org/10.1038/s41598-023-30028-w

Azeez, N.A., Odufuwa, O.E., Misra, S., Oluranti, J., & Damaševičius, R. (2021). Windows PE malware detection using ensemble learning. *Informatics*, *8*(1), 10. https://doi.org/10.3390/informatics8010010

Baghirov, E. (2023a). Malware detection based on opcode frequency. *Journal of Applied Information Technology*, *1*(1), 3–7. https://jpit.az/uploads/article/en/2023_1/MALWARE_DETECTION_BASED_ON_OPCODE_FREQUENCY.pdf

Baghirov, E. (2025). A comprehensive investigation into robust malware detection with explainable AI. *Cyber Security and Applications*, *3*, 100072. https://doi.org/10.1016/j.csa.2024.100072

Belaoued, M., Mazouzi, S. (2016). A Chi-Square-based decision for real-time malware detection using PE-file features. *Journal of Information Processing Systems*, *12*(4), 644–660.

Catak, F.O., Yazı, A.F., Elezaj, O., & Ahmed, J. (2020). Deep learning-based sequential model for malware analysis using Windows EXE API calls. *PeerJ Computer Science*, *6*, e285. https://doi.org/10.7717/peerj-cs.285

Chen, Y.-C., Chen, H.-Y., Takahashi, T., Sun, B., & Lin, T.-N. (2021). Impact of code deobfuscation and feature interaction in Android malware detection. *IEEE Access*, *9*. Retrieved from https://ieeexplore.ieee.org/abstract/document/9529344/

Erfina, A., Hidayat, R., Rizaldi, R. R., Puput, Maulana, R., & Sembiring, F. (2023). Analyzing code injection attacks on applications of Android devices and emulator. *2023 IEEE 9th International Conference on Computing, Engineering and Design (ICCED)*, 1–6.

Huang, Y., Zhang, J., Shan, Z., & He, J. (2024). Compression represents intelligence linearly. *arXiv preprint arXiv:2404.09937*.

Huidobro, C.B., Vidal, D.C., Cubillos, C., Palma Esquivel, M., & Cabrera-Paniagua, D. (2017). Obfuscation-based malware update: A comparison of manual and automated methods. *International Journal of Computer Communications and Control*, *12*, 461–474.

Imran, M., Appice, A., & Malerba, D. (2024). Evaluating realistic adversarial attacks against machine learning models for Windows PE malware detection. *Future Internet*, *16*(5), 168. https://doi.org/10.3390/fi16050168

Islam, T., Rahman, S.S.M., Hasan, M., Sayed, A., Rahaman, A.S.M.M., & Jabiullah, M.I. (2020). Evaluation of N-Gram based multi-layer approach to detect malware in Android. *Procedia Computer Science*, **171**, 1074–1082. DOI: 10.1016/j.procs.2020.04.115.

Jamar, R., Sogani, A., Mudgal, S., Bhadra, Y., & Churi, P. (2017). E-shield: Detection and prevention of website attacks. *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 706–710.

Li, N. (2009). Data encryption. In *Encyclopedia of Database Systems* (p. 574). Springer US. https://doi.org/10.1007/978-0-387-39940-9_98

Lundberg, S.M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*.

Malwarebytes Labs. (2024). State of Malware 2024: What consumers need to know. Retrieved from https://www.malwarebytes.com/blog/personal/2024/02/state-of-malware-2024-what-consumers-need-to-know

Shaukat, K., Luo, S., & Varadharajan, V. (2024). A novel machine learning approach for detecting first-time-appeared malware. *Engineering Applications of Artificial Intelligence*, *131*, 107801. https://doi.org/10.1016/j.engappai.2023.107801

Microsoft Build contributors. (2024). PE format - Win32 apps. Retrieved from https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#general-concepts

Mirzazadeh, R., Moattar, M.H., & Jahan, M.V. (2015). Metamorphic malware detection using linear discriminant analysis and graph similarity. *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*, 61–66.

Mbunge, E., Muchemwa, B., Batani, J., & Mbuyisa, N. (2023). A review of deep learning models to detect malware in Android applications. *Cyber Security and Applications*, *1*, 100014. https://doi.org/10.1016/j.csa.2023.100014

Ribeiro, M.T., Singh, S., & Guestrin, C. (2016). 'Why should I trust you?": Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://arxiv.org/abs/1602.04938

Selamat, N.S., Mohd Ali, F.H., & Abu Othman, N.A. (2016). Polymorphic malware detection. *2016 6th International Conference on IT Convergence and Security (ICITCS)*, 1–5.

Shukla, H., Patil, S., Solanki, D., Singh, L., Swarnkar, M., & Thakkar, H. K. (2019). On the design of supervised binary classifiers for malware detection using portable executable files. *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, 141–146. DOI: 10.1109/IACC48062.2019.8971519.

Youssef, E.S., Labeau, F., Kassouf, M., & Alarie, S. (2022). Cyberattacks against direct load control of residential electric water heaters in smart grids. *2022 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 1–5.

Yousuf, I. (2023). Windows malware detection dataset. *Figshare Dataset*. https://doi.org/10.6084/m9.figshare.21608262.v1

You, I., Yim, K. (2010). Malware obfuscation techniques: A brief survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, 297–300.