Refining Malware Analysis with Enhanced Machine Learning Algorithms using Hyperparameter Tuning

[Do not insert author name or contact details now - only to be completed after acceptance]

Abstract: Many researchers address challenges and limitations inherent to machine learning algorithms to optimize classifier performance. Overfitting, a prevalent issue, arises when models are excessively complex and trained on noisy data, leading to suboptimal generalization to new data. Another concern is underfitting, where models are overly simplistic and fail to capture data complexity. This comprehensive investigation focuses on machine learning's application to malware classification, specifically targeting PE files. The study addresses these limitations using ensemble methods and pre-processing techniques, including feature selection and hyperparameter tuning. The primary objective is to augment classifier performance. Through a comparative study that aims to classify PE files as malicious or benign through analysis of machine learning methodologies such as random forests, decision trees, and gradient boosting, the study highlights the superiority of the random forests algorithm, achieving a remarkable 99% accuracy rate. Thoroughly assessing the strengths and limitations of each algorithm provides valuable insights into effectively handling diverse malware categories. This paper underscores the significance of ensemble methods, feature engineering, and pre-processing in enhancing classifier performance.

Keywords: Malware Analysis, Malware Detection, Machine Learning, Optimization, Hyperparameter Tunning, Data Balancing, Feature Selection

1 Introduction

The rapid proliferation of malware poses significant challenges for traditional signature-based detection systems. Machine learning algorithms offer a promising solution by leveraging the power of computational intelligence to analyze large volumes of data and detect previously unseen malware variants [1] [2]. However, the success of machine learning in malware analysis depends on several factors, including the quality of the dataset, feature selection, algorithm choice, and hyperparameter tuning [3] [4].

Previous studies have highlighted the importance of feature selection in reducing the dimensionality of the data and improving the performance of machine learning models [5] [6]. In the context of malware analysis, selecting the most relevant features from a large pool is critical, considering the complexities and imbalances in malware datasets [7]. Additionally, choosing the appropriate machine learning algorithm and optimizing its hyperparameters significantly impact the accuracy and efficiency of the classification process.

In this research, an in-depth investigation has been conducted on the application of machine learning for malware classification, with a specific focus on Portable Executable (PE) files [8]. The dataset used in this research contains 96765 malware samples and 41323 legitimate samples, presenting significant imbalances and complexities in the number of features. To address this challenge, feature selection techniques were used to specify the most relevant features among 54 and used them to train and test 5 machine learning algorithms, including random forests, decision trees, and gradient boosting. Choosing the best hyperparameters for each algorithm significantly impacted the accuracy of models and improved model performance by performing Hyperparameter tuning. These algorithms were evaluated based on their ability to classify PE files as malicious or benign accurately, and we analyzed the results in terms of accuracy, precision, and performance facing unseen files.

The primary contribution of this research is twofold. Firstly, we conducted an in-depth investigation into the application of machine learning for malware classification, specifically targeting PE files. We analyzed a large dataset comprising 96,765 malware samples and 41,323 legitimate samples, considering the complexities and imbalances in the number of features. Secondly, feature selection techniques were employed to identify the most relevant features and trained and tested four machine learning algorithms, including random forests, decision trees, and gradient boosting. By optimizing hyperparameters for each algorithm through hyperparameter tuning, we aimed to improve classification accuracy and efficiency.

By evaluating the performance of these algorithms in terms of accuracy, precision, and performance on unseen files, the importance of effective pre-processing techniques [9] emphasizes, such as feature selection and hyperparameter tuning. Furthermore, this research serves as a valuable resource and reference for future initiatives in the field of malware detection, contributing to advancing machine learning algorithms and highlighting areas that require further research.

This research aims to thoroughly examine the use of machine learning in malware detection to gain a deeper understanding of its potential and limitations. Specifically, the study will address the following key questions:

- How effective is our proposed machine learning technique in malware detection compared to other commonly used machine learning methods?
- To what extent can data balancing techniques improve the accuracy and robustness of machine learning models in malware detection?
- How does the application of data pre-processing and feature selection techniques impact the performance of machine learning algorithms in detecting malware?
- How can hyperparameter tuning be used to optimize the performance of machine learning algorithms in malware detection?

Malware detection is a challenging task in cybersecurity, and various approaches have been proposed to tackle this issue. Researchers have recently explored different methods, such as machine learning algorithms, signature-based protection [10], heuristic-based methods [11], and behaviour-based methods [12]. However, previous surveys have limitations regarding coverage of new research trends and the taxonomy of feature types used for malware detection.

Yi-Hsien Chen. [13] focuses on automated malware analysis to address the evolving threat of malicious binaries. The researchers developed Malware Expert, an expert system based on a graph neural network, to identify essential functions and visualize relationships within analyzed samples. The system achieved competitive detection performance and provided intuitive explanations of its predictions. Comparative analysis with expert-made reports showed accurate identification of essential functions. The study represents a significant step towards automated program behavior analysis and provides valuable insights for security analysts and researchers. Karampudi [14] conducted the study, focus is highlighting the limitations of static techniques, such as the decision tree algorithm, in detecting malware. A hybrid approach is proposed to overcome these constraints, integrating multiple machine learning (ML) algorithms using ensemble learning techniques. The Ada Boost classifier is implemented to enhance the efficiency of the hybrid model. The study evaluates the performance of various ML algorithms, including Decision Tree, Gaussian Na¨ıve Bayes, Random Forest, and Linear SVM, for analyzing malware content. Algorithms with lower accuracy are boosted using Ada Boost to improve performance, resulting in significant accuracy improvements. The study concludes that the hybrid approach, combined with ensemble learning, yields more effective and accurate results in malware detection.

Jin Li. [15] focus on addressing the alarming growth rate of malicious apps in the mobile ecosystem, particularly on the Android platform. The authors introduce SigPID, a malware detection system based on permission usage analysis. SigPID utilizes a three-level pruning approach to identify the most significant permissions for distinguishing between benign and malicious apps. Machine-learning-based classification methods are employed to classify different families of malware and benign apps. The evaluation shows that using only 22 significant permissions, SigPID achieves high precision, recall, accuracy, and F-measure, similar to the baseline approach analyzing all permissions. Moreover, SigPID demonstrates superior effectiveness to other state-of-the-art approaches, detecting a high percentage of malware in the dataset, including unknown/new malware samples. Kouliaridis. [16] focus on using machine learning techniques for malware analysis in Windows environments, specifically for Portable Executables (PEs). The authors review and systematize existing literature, categorizing papers based on their objectives, the features they use, and the machine learning algorithms employed. They identify issues and challenges, such as anti-analysis techniques, selection of operations for analysis, and dataset availability. The article proposes a taxonomy to synthesize the state of the art and highlights new research directions and topical trends. It also introduces the concept of malware analysis economics, considering the trade-offs between analysis accuracy, time, and cost. The study provides a comprehensive overview and guides security analysts interested in applying machine learning to automate malware analysis operations.

The present paper aims to devise an effective malware detection system using machine learning techniques. The study's objectives highlight the pivotal roles of data pre-processing, feature selection, and model selection in achieving precise outcomes. By currently emphasizing the significance of techniques like data balancing and engineering to counter biases, the research is contributing to a balanced model performance. Furthermore, the study aims to identify the most proficient machine learning algorithm for malware detection, with the Random Forest model currently emerging as a leading contender. Lastly, the research is currently introducing a novel perspective by uncovering optimal initial hyperparameter values, intended to serve as a reference for future pursuits in malware detection.

The paper is organized as follows: Section 2 is divided into five sub-sections: Data Collection, outlining the process of gathering data related to malware samples; Feature Selection and Pre-Processing, detailing steps taken to enhance data quality; Data Balancing Techniques, addressing the mitigation of class imbalance : Model Implementation and Hyperparameter Tuning, explaining the implementation process and its optimization through hyperparameters; Model Evaluation, systematically assessing the performance of different algorithms. Section 3 presents the outcomes of the analysis and engages in a comprehensive discourse about the strengths and limitations of the implemented models. Finally, in Section 4, the Conclusion highlights the research's core findings, contributions to the field of malware analysis, and recommendations for future research directions.

2. METHODS

2.1. Data Collection

In this research, a robust and efficient data processing and model optimization workflow has been implemented to address this project's specific challenges and requirements. The approach incorporates advanced techniques tailored to tackle the issues associated with unbalanced data and a large, diverse feature set, which can often result in complex models prone to over-fitting.

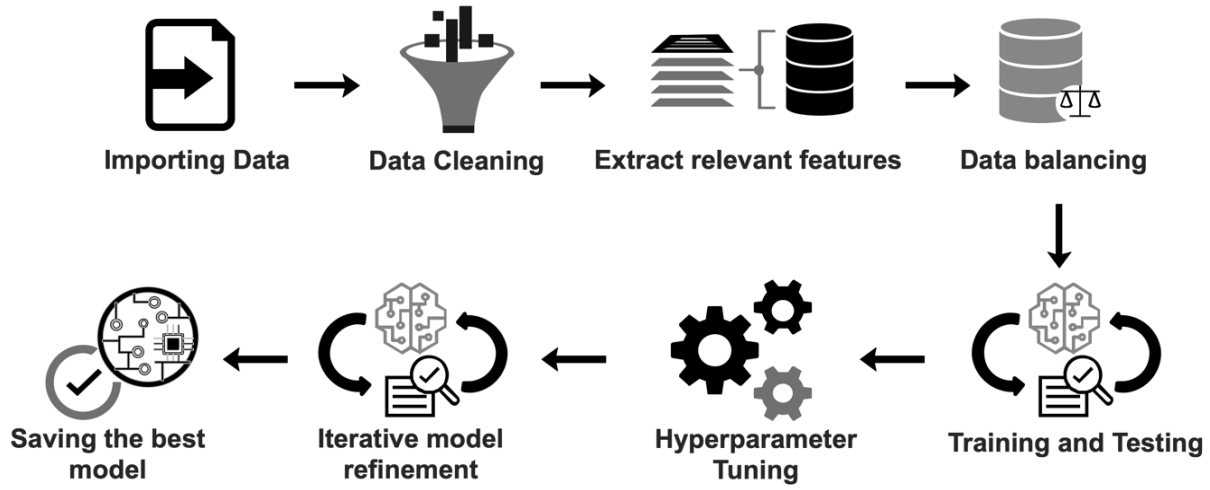

Figure 1. Data Processing and Model Optimization Workflow.

The creation of a high-quality data set is crucial for generating accurate results. To achieve this, a substantial number of both legitimate and malicious files are required. The malicious and legitimate files used in thisstudy was obtained from the Kaggle[17] website. Additionally, for testing the efficiency of the developed models, a diverse range of legitimate and malicious files were downloaded from various websites on the Internet, VirusShare [15] Malware Bazzar [19], and VirusTotal [20]. To evaluate the effectiveness of machine learning algorithms in malware detection, a comprehensive dataset comprising 138,042 instances with 57 features was utilized in this study. This dataset comprised 96,000 malicious files and 41,000 legitimate files, representing a distribution reflecting real-world malware samples. Several essential steps were undertaken to assess the machine learning algorithms' performance.

Firstly, a data balancing technique was applied to address the class imbalance issue. This involved adjusting the number of instances in each class to ensure a more representative distribution, thereby mitigating potential bias in the model training process.

Next, feature selection was performed to identify the most relevant attributes for the malware classification task. By determining the discriminative features, the dimensionality of the dataset was reduced, enhancing the efficiency and effectiveness of the subsequent machine learning algorithms.

To optimize the performance of the models, hyperparameter tuning was conducted. This process involved systematically adjusting the algorithm-specific parameters to identify the optimal configuration that yields the highest classification accuracy and overall model performance.

Each machine-learning algorithm was trained and evaluated using the prepared dataset following the pre-processing steps. The performance metrics such as accuracy, precision, recall, and F1-score were computed to assess the effectiveness of each algorithm in accurately identifying and classifying malware instances.

The trained models were saved as files for later use to ensure practical usability. This allows for easy deployment and application in real-world scenarios, where new and unseen data can be processed efficiently.
A separate set of PE files was used to evaluate the models' performance on unseen data. These PE files were subjected to feature extraction, followed by applying the pre-saved models. The results obtained from this evaluation process provided insights into the effectiveness and generalization capability of the models.

By systematically implementing these steps and evaluating the performance of the machine learning algorithms on both the training and unseen data, this study aimed to improve the accuracy and efficiency of malware detection. The findings and methodologies presented in this research contribute to advancing the field of malware analysis and provide valuable insights for future research and practical applications in cybersecurity.

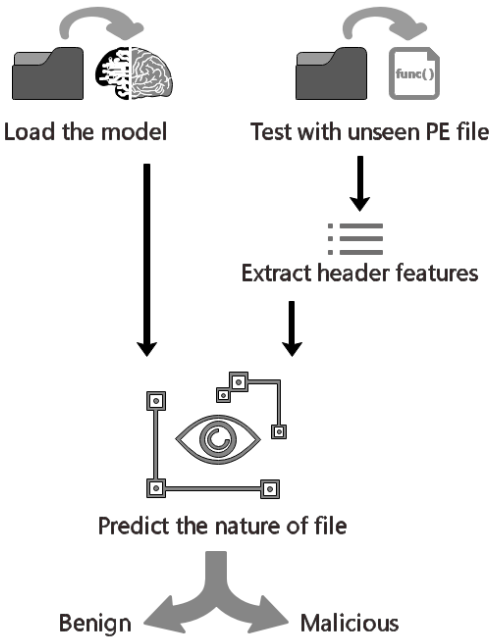

Figure 2. Advanced Architecture for Accurate Classification of Unseen Files.

2.2. Data Balancing and Feature Selection

This step ensures that the data used for training the model is balanced, meaning each class has an equal number of samples. This is important because imbalanced data can lead to a biased model toward the majority. The graphs below show the percentage of samples in each class before and after balancing the data:

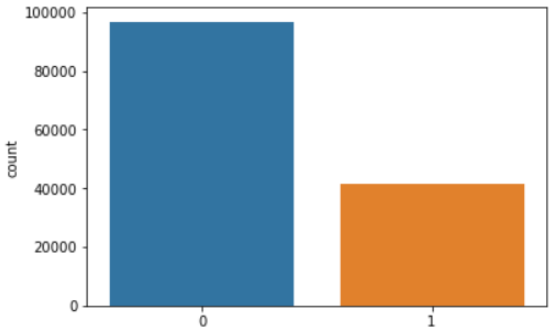

Figure 3. Before Balancing.

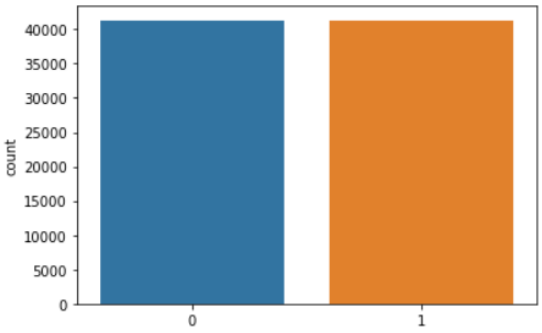

Figure 3. After Balancing.

Then, identifying the most relevant and significant features is crucial for classifying files as malicious or legitimate. With 54 columns available, it is important to carefully select those that will provide the most informative insights for the classification process. If you don't use feature selection for a dataset with 54 columns, you run the risk of overfitting, decreased model interpretation ability and increased computational cost. With so many features, some of them hurt the target variable and may even have a negative impact on the model's performance. Additionally, having a large number of features can lead to a high dimensionality, making it difficult for the model to find a meaningful pattern in the data.

Furthermore, the increased computational cost can result in more extended training and prediction times. Feature selection is important in this scenario as it helps reduce the model's complexity and improve its performance by only

selecting the most relevant and meaningful features. ExtraTreesClassifier was utilized to aid in the classification of files.

The algorithm helped identify the most important features for accurate classification by analyzing many potential features. As a result, 14 features were pinpointed as being the most useful for determining whether a file was malicious. By using these 14 features, the classification task achieved high accuracy. This highlights the effectiveness of ExtraTreesClassifier in selecting the required features for classifying files.

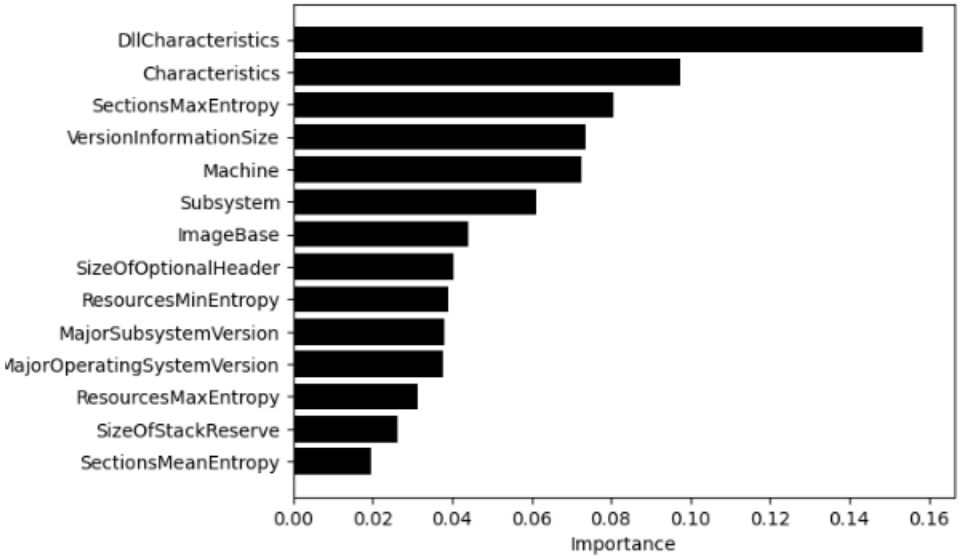

Figure 5. Importance of the relevant feature.

2.3. Model implementation through Hyperparameter tuning

To implement the chosen models, the default settings have been utilized as recommended by the software package for each of the models used. These models include DecisionTree, RandomForest, Adaboost, GradientBoosting, and GNB. Based on previous research papers, this approach was utilized to test the capabilities of algorithms without external guidance to provide consistency and minimize potential sources of bias [18] or error in the analysis. Next, a Hyper-parameter setting approach will be used to determine suitable parameters.

Hyperparameter tuning is an essential step in improving the performance of machine learning models. In this study, a GridSearchCV method from the Scikit-Learn library was used to tune the hyperparameters of the models. GridSearchCV performs an exhaustive search over a specified parameter grid to find the optimal combination of hyperparameters.

A list of models was created, each with a different set of hyperparameters to be tuned. The models included DecisionTree, RandomForest, Adaboost, GradientBoosting, and GNB. For each model, a grid of hyperparameters was defined and used as input for the GridSearchCV function. The GridSearchCV function was run with a 5-fold cross-validation and n jobs = -1, which allowed the function to use all available cores on the machine to speed up the computation process. After running the function, the best parameters for each model were extracted and used to train a new model. The new models were then used to predict the labels of the test set, and the accuracy of each model was calculated using the accuracy score function from Scikit-Learn.

The recommended hyperparameters used to test the model's accuracy are widely used and have produced good results in many previous studies. For example, the 'max depth' parameter controls the maximum depth of the decision tree, and it is known that a deeper tree can result in overfitting. Therefore, the values 5, 10, and 15 are used to avoid overfitting and still capture the complexity of the data. Similarly, 'min samples leaf' is used to control the minimum number of samples required to form a leaf node, and values of 1, 5, and 10 are chosen to ensure a sufficient number of samples in each leaf. For the Random Forest model, parameters such as 'n estimators', 'max depth', 'max features', and 'min samples split' were tuned to balance bias and variance. Finally, for Adaboost and GradientBoosting models, we used 'n estimators' and 'learning rate' parameters to control the number of weak classifiers and the contribution of each classifier to the final prediction.

Overall, we chose these values based on their proven effectiveness and the specific characteristics of our dataset. At last, the accuracy of the new models was compared to the accuracy of the models with default hyperparameters. This allowed for an evaluation of whether the hyperparameter tuning process improved the performance of the models.

2.4. Model Evaluation

This study conducted experiments on a computer with an Intel Core i7 Central Processing Unit (CPU) with 16 Gigabytes (GB) of Random-Access Memory (RAM) and an Intel® UHD Graphics processing unit (GPU) with 4.1 Gigabytes of dedicated video memory. The operating system used was Windows 10. The experiments were implemented using the Python programming language, version 3.8.5, and various libraries including NumPy, Pandas, Scikit-learn, PeFile, and TensorFlow. The data used in the experiments were stored locally as files.

2.4.1. Training and evaluating each algorithm

Once the key attributes have been established, the next step is to train and evaluate different classification algorithms using these attributes. This can be done by dividing the data into a training set and a test set and using the training set to train the model and the test set to evaluate the model's performance. The data division was done to ensure that the accuracy of the machine learning algorithms could be evaluated. The training set consisted of "X train" and "y train". "X train" contains all the feature values for each row, excluding the label column, while "y train" contains only the label values for each row. In the experiment, five machine learning algorithms were employed once the data was divided, including Random Forest, Decision Tree, Adaboost, Gradient Boosting, and Gaussian Naive Bayes (GNB). The trained models were then used to classify the data in the test set "X test" and "y test" and evaluate their accuracy. The results of this evaluation provide insights into the efficiency and performance of each algorithm.

2.4.2. Training and evaluating each algorithm

Based on the majority of methods employed by previous researchers, it is recommended to follow a specific sequence of steps. First, train and evaluate the algorithms [22]. Once this process is completed, it is crucial to determine the most efficient model by calculating various metrics such as accuracy, false positives, and false negatives.

Finally, save the model with the highest efficiency as a file for future use. Each algorithm's efficiency is analyzed through various metrics, such as accuracy, false positive rate, and false negative rate. The model's accuracy is evaluated by utilizing the score function from the Scikit library in Python. Calculating false positive and false negative rates is done using the confusion matrix, sometimes called the error matrix. The Confusion Matrix is a table that summarizes the performance of a binary classification algorithm. This matrix compares the predicted values with the actual values, as shown in Table 1 below:

Table 1. Confusion Matrix

	Actual Positive (1)	**Actual Negative (0)**
Predicted Positive (0)	True Positive	False Positive
Predicted Negative (1)	False Negative	True Negative

- True Positives (TP): The number of correctly predicted instances as positive.
- False Positives (FP): The number of instances predicted as positive but were actually negative.
- True Negatives (TN): The number of correctly predicted instances as negative.
- False Negatives (FN): The number of instances predicted as negative but were actually positive.

The metrics TPR, TNR, precision, and accuracy can be calculated using the values in the confusion matrix:

True Positive Rate: Sensitivity or Recall is the fraction of actual positive instances that were correctly classified as positive.

$$\boldsymbol{TPR} = \left(\frac{\boldsymbol{TP}}{\boldsymbol{FN}}\right)$$

False Positive Rate: also known as Specificity, is the fraction of actual negative instances that were incorrectly classified as positive

$$FPR = \left(\frac{FP}{FP + TN} \right)$$

Precision: the fraction of positive predictions that were positive

$$Precision = \left(\frac{TP}{TP + FP} \right)$$

Accuracy: the fraction of instances that were correctly classified by the algorithm

$$Accuracy = \left(\frac{TP}{TP + FP} \right)$$

In conclusion, TPR and TNR represent the ability of the algorithm to predict positive and negative cases, respectively correctly. Precision measures the ability of the algorithm to make accurate positive predictions, while accuracy is the overall measure of the algorithm's performance.

2.5. Performance Testing

Finally, a technique extracts features from new files and classifies them as malicious or benign. By parsing the PE file headers, key attributes are gathered and stored in a Python dictionary, with the attribute name serving as the key and its content as the value of the dictionary. These extracted features are then passed to a pre-trained machine learning model, saved as a .pkl file, and loaded using joblib.load.

The performance of the saved model can be tested on unseen data by extracting features from the PE file and reading the pre-saved model to predict the class label. The models have been rigorously tested on unseen data to ensure their accuracy and reliability. In this study, the methodology and framework utilized were designed to help achieve high accuracy in classifying malicious and legitimate files. The effectiveness of ExtraTreesClassifier in selecting the required features was highlighted, as well as the importance of balancing data to avoid bias in the training process.

3. RESULTS AND DISCUSSION

3.1. Algorithms Performance

Table 2 below displays the values of performance criteria for every algorithm that was tested:

Table 2. Before Tunning

Algorithm	Accuracy	Precision	True positive Rate (%)	False Negative Rate (%)
DecisionTree	0.99062311	0.988	0.992	0.012
RandomForest	0.99165154	0.992	0.995	0.008
Adaboost	0.98324258	0.984	0.982	0.016
GradientBoosting	0.98505747	0.987	0.987	0.013
GNB	0.49975801	1.000	0.000	0.000

3.2. Optimized Hyperparameters for Various Models

3.2.1. Algorithms Performance

In this section, the hyperparameters chosen presented for each model were carefully selected to ensure a balance between bias and variance. Specific values were chosen for each hyperparameter to avoid overfitting and capture the complexity of the data. Table 3, summarizing the chosen hyperparameters, is also presented.

Table 3. Hyperparameters Chosen for Each Model

Model	The Recommended Hyperparameters Before Tunning
DecisionTree	{”max depth”: [5, 10, 15], ”min samples leaf”: [1, 5, 10]}
RandomForest	{”max depth”: [5, 10, 15], ”max features”: [5, 10, 15], ”min samples split”: [2, 5, 10]}
Adaboost	{”n estimators”: [5, 10, 15], ”learning rate”: [0.1, 0.5, 1.0]}
GradientBoosting	{”n estimators”: [5, 10, 15], ”learning rate”: [0.1, 0.5, 1.0]}
GNB	{}

3.2.2. Algorithms Performance

In table 4 below, we provide a summary of the best hyperparameters for each model after tuning:

Table 4. Best Hyperparameters for Each Model

Model	The Recommended Hyperparameters Before Tunning
DecisionTree	{”max depth”: [5, 10, 15], ”min samples leaf”: [1, 5, 10]}
RandomForest	{”max depth”: [5, 10, 15], ”max features”: [5, 10, 15], ”min samples split”: [2, 5, 10]}
Adaboost	{”n estimators”: [5, 10, 15], ”learning rate”: [0.1, 0.5, 1.0]}
GradientBoosting	{”n estimators”: [5, 10, 15], ”learning rate”: [0.1, 0.5, 1.0]}
GNB	{}

Table 5 displays the values of performance criteria for every algorithm after tunning:

Table 5. After Tunning

Algorithm	Accuracy	Precision	True positive Rate (%)	False Negative Rate (%)
DecisionTree	0.994007	0.988	0.992	0.012
RandomForest	0.999143	0.992	0.995	0.008
Adaboost	0.997767	0.984	0.982	0.016
GradientBoosting	0.990260	0.987	0.987	0.013
GNB	0.504597	1.000	0.000	0.000

3.3. Discussion

Before tuning, the algorithms' accuracies ranged from 49% to 99%. After that, the hyperparameters of each algorithm were tuned to obtain accuracies that align with the good performance of algorithms. The recommended hyperparameters for each algorithm were chosen based on the recommended values commonly used in such a similar type of data and problematics. After tuning, the accuracy of the algorithms improved except for the Gaussian Naive Bayes (GNB) algorithm, which had an accuracy of only 50%. This is not surprising since GNB is a simple algorithm that assumes that the features are independent, which is not the case in the real world. Therefore, it is not suitable for complex problems like malware detection.

After tuning the hyperparameters ’max depth’ and ’min samples leaf’ for the Decision Tree algorithm, the accuracy improved from 99.06% to 99.40%. The optimal

depth was 10, and a leaf node with only one sample was enough to make the prediction. However, overfitting was possible as the model was too complex and captured noise in the training data. The'max depth' and 'min samples leaf' hyperparameters could have been reduced to reduce overfitting. However, this could also decrease the model's ability to capture nuances in the data, leading to a slight decrease in accuracy. Despite this, the accuracy drop was relatively small, but the tuned model did not perform well, with an accuracy of 99.06%.

The recommended hyperparameters for the Random Forest algorithm were 'max depth', 'max features', and 'min samples split'. After tuning, 'max depth' was set to 15, 'max features' was set to 5, and 'min samples split' was set to 2. Additionally, 'n estimators' was set to 15, which was not in the recommended hyperparameters. The accuracy improved from 99.16% to 99.91%. This means that increasing the depth of the trees, reducing the number of features, and reducing the minimum number of samples required to split a node, while increasing the number of trees, lead to better performance.

For the Adaboost algorithm, the recommended hyperparameters were "n_estimators" and "learning rate". After tuning, 'n estimators' was set to 15, and 'learning rate' was set to 0.5. The accuracy decreased from 99.32% to 99.77%. This means that the algorithm is very sensitive to the hyperparameter learning rate, and a learning rate 0.5 was too high for this dataset.

The recommended hyperparameters for the Gradient Boosting algorithm were 'n estimators' and 'learning rate'. After tuning, 'n estimators' was set to 15, and 'learning rate' was set to 1. The accuracy improved from 98.32% to 99.77%. This means that a high learning rate was suitable for this algorithm. A continuous testing methodology against new files was adopted to evaluate the performance of machine learning models on unseen files. The methodology involved testing the models after every phase of optimization, which included data balancing and tuning hyperparameters. The results showed that the models could maintain their performance over time, with a slight increase in performance observed after each phase. This suggests that the continuous testing approach is an effective way to evaluate the performance of machine learning models on unseen files.

Overall, the research presents valuable perspectives on using machine-learning algorithms to detect file malware. The study emphasizes the importance of selecting suitable algorithms and fine-tuning hyperparameters to attain optimal performance while also considering factors such as accuracy and speed.

4. CONCLUSION

In conclusion, this research aimed to develop an effective malware detection system using machine learning techniques. The study highlights the importance of data pre-processing, feature selection, and model selection in achieving accurate results. It was found that data balancing and data engineering techniques are necessary to avoid bias in the model towards one class over the other.

The Random Forest model was found to have the best accuracy and performance in detecting malware. However, it is important to note that the optimal hyperparameters for each algorithm can vary depending on the dataset and algorithm used. Therefore, it is recommended to tune the hyperparameters for each problem separately.

The novelty of this research lies in discovering the best initial hyperparameter values for each model that can be used as a guideline for future machine learning research and malware detection actors. It was also found that high accuracy alone cannot ensure effective detection and prevention of malicious software. Therefore, evaluating the models against new unseen files is recommended to ensure better performance. In summary, this research provides a valuable contribution to the field of malware detection by proposing effective techniques for data pre-processing, feature selection, and model selection. The findings of this study can be used as a guideline for future research in this field, and the novel discovery of the best initial hyperparameters can save time and effort in finding optimal values for each algorithm.

Despite the promising results achieved by the proposed model, there are still some areas for future improvements. One potential direction is incorporating more advanced deep learning techniques, such as attention mechanisms and reinforcement learning, to enhance the model's ability to capture complex patterns and relationships in malware files. This approach will provide an increase in the performance of models over time when facing new files and will guarantee continuous learning. Another area for future exploration is to evaluate the model's performance on different operating systems and file formats, as the current study mainly focused on Windows executable files. Finally, as with any machine learning model, ongoing [20] g monitoring and evaluating the model's performance is necessary to ensure its continued effectiveness over time.

REFERENCES

[1] H. K. Jabbar and R. Z. Khan, "Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study)," *Computer Science, Communication and Instrumentation Devices*, 2014, Accessed: Aug. 04, 2023. [Online]. Available: https://www.academia.edu/12972082/METHODS_TO_AVOID_OVER_FITTING_AND_UNDER_FITTING_IN_SUPERVISED_MACHINE_LEARNING_COMPARATIVE_STUDY_

[2] M. Decuyper, M. Stockhoff, S. Vandenberghe, al -, and X. Ying, "An Overview of Overfitting and its Solutions," *J Phys Conf Ser*, vol. 1168, no. 2, p. 022022, Feb. 2019, doi: 10.1088/1742-6596/1168/2/022022.

[3] H. J. P. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of Tuning Hyperparameters of Machine Learning Algorithms," Jul. 2020, Accessed: Aug. 04,

2023. [Online]. Available: https://arxiv.org/abs/2007.07588v1

[4] S. Symeonidis, D. Effrosynidis, and A. Arampatzis, "A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis," *Expert Syst Appl*, vol. 110, pp. 298–310, Nov. 2018, doi: 10.1016/J.ESWA.2018.06.022.

[5] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artif Intell*, vol. 97, no. 1–2, pp. 245–271, 1997, doi: 10.1016/S0004-3702(97)00063-5.

[6] A. M. Radwan, "Machine learning techniques to detect maliciousness of portable executable files," *Proceedings - 2019 International Conference on Promising Electronic Technologies, ICPET 2019*, pp. 86–90, Oct. 2019, doi: 10.1109/ICPET.2019.00023.

[7] H. S. Obaid, S. A. Dheyab, and S. S. Sabry, "The impact of data pre-processing techniques and dimensionality reduction on the accuracy of machine learning," *IEMECON 2019 - 9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference*, pp. 279–283, Mar. 2019, doi: 10.1109/IEMECONX.2019.8877011.

[8] S. Pothuganti, "International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Review on over-fitting and under-fitting problems in Machine Learning and solutions," 2018, doi: 10.15662/IJAREEIE.2018.0709015.

[9] Z. Manzoor, M. Tech Scholar, and N. K. Sandhu, "HYBRID FIREFLY-ANFIS ALGORITHM TO DETECT MALICIOUS SOFTWARE," *Dogo Rangsang Research Journal UGC Care Group I Journal*, vol. 12, 2022.

[10] M. Shoaib Akhtar and M. S. Akhtar, "Analyzing and comparing the effectiveness of various machine learning algorithms for Android malware detection," *Advances in Mobile Learning Educational Research*, vol. 3, no. 1, pp. 570–578, Dec. 2023, doi: 10.25082/AMLER.2023.01.005.

[11] Y. H. Chen, S. C. Lin, S. C. Huang, C. L. Lei, and C. Y. Huang, "Guided Malware Sample Analysis based on Graph Neural Networks," *IEEE Transactions on Information Forensics and Security*, 2023, doi: 10.1109/TIFS.2023.3283913.

[12] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," *IEEE Trans Industr Inform*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018, doi: 10.1109/TII.2017.2789219.

[13] V. Kouliaridis and G. Kambourakis, "A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection," *Information 2021, Vol. 12, Page 185*, vol. 12, no. 5, p. 185, Apr. 2021, doi: 10.3390/INFO12050185.

[14] "Kaggle: Your Machine Learning and Data Science Community." https://www.kaggle.com/ (accessed Aug. 04, 2023).

[15] "VirusShare.com." https://www.virusshare.com/ (accessed Aug. 04, 2023).

[16] "MalwareBazaar | Malware sample exchange." https://bazaar.abuse.ch/ (accessed Aug. 04, 2023).

[17] "VirusTotal - Home." https://www.virustotal.com/gui/home/upload (accessed Aug. 04, 2023).

[18] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A Survey on Bias and Fairness in Machine Learning," *ACM Comput Surv*, vol. 54, no. 6, Aug. 2019, doi: 10.1145/3457607.

[19] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *International Journal of Robotics Research*, vol. 40, no. 4–5, pp. 698–721, Apr. 2021, doi: 10.1177/0278364920987859/ASSET/IMAGES/LARGE/10.1177_0278364920987859-FIG3.JPEG.

[20] R. Patil and W. Deng, "Malware analysis using machine learning and deep learning techniques," *Conference Proceedings - IEEE SOUTHEASTCON*, vol. 2, Mar. 2020, doi: 10.1109/SOUTHEASTCON44009.2020.9368268.