algorithms

MDPI

Article

Tree-Based Classifier Ensembles for PE Malware Analysis: A Performance Revisit

Maya Hilda Lestari Louk [1] and Bayu Adhi Tama [2,*]

[1] Department of Informatics Engineering, University of Surabaya, Surabaya 60293, Indonesia
[2] Department of Information Systems, University of Maryland, Baltimore County (UMBC), Baltimore, MD 21250, USA
* Correspondence: bayu@umbc.edu

Abstract: Given their escalating number and variety, combating malware is becoming increasingly strenuous. Machine learning techniques are often used in the literature to automatically discover the models and patterns behind such challenges and create solutions that can maintain the rapid pace at which malware evolves. This article compares various tree-based ensemble learning methods that have been proposed in the analysis of PE malware. A tree-based ensemble is an unconventional learning paradigm that constructs and combines a collection of base learners (e.g., decision trees), as opposed to the conventional learning paradigm, which aims to construct individual learners from training data. Several tree-based ensemble techniques, such as random forest, XGBoost, CatBoost, GBM, and LightGBM, are taken into consideration and are appraised using different performance measures, such as accuracy, MCC, precision, recall, AUC, and F1. In addition, the experiment includes many public datasets, such as BODMAS, Kaggle, and CIC-MalMem-2022, to demonstrate the generalizability of the classifiers in a variety of contexts. Based on the test findings, all tree-based ensembles performed well, and performance differences between algorithms are not statistically significant, particularly when their respective hyperparameters are appropriately configured. The proposed tree-based ensemble techniques also outperformed other, similar PE malware detectors that have been published in recent years.

Keywords: portable executable malware; tree-based ensemble; performance comparison; statistical significance test

check for updates

Citation: Louk, M.H.L.; Tama, B.A. Tree-Based Classifier Ensembles for PE Malware Analysis: A Performance Revisit. *Algorithms* **2022**, *15*, 332. https://doi.org/10.3390/a15090332

Academic Editors: Francesco Bergadano and Giorgio Giacinto

Received: 29 August 2022
Accepted: 14 September 2022
Published: 17 September 2022

1. Introduction

Malware (e.g., malicious software) is commonly recognized as one of the most potent cyber threats and hazards to modern computer systems [1,2]. It is an overarching word that refers to any code that potentially has a destructive, harmful effect [3]. On the basis of their behavior and execution processes, malicious softwares are categorized as worms, viruses, Trojan horses, rootkits, backdoors, spyware, logic bombs, adware, and ransomware. Computer systems are hacked for a variety of reasons, including the destruction of computer resources, financial gain, the theft of private and confidential information and the use of computing resources, as well as the inaccessibility of system services, to name a few [4].

Malware is recognized using signature-based or behavior-based methods. The signature-based malware detection techniques are quick and effective, but obfuscated malware can quickly circumvent them. In contrast, behavior-based methods are more resistant to obfuscation. Nonetheless, behavior-based methods are relatively time-intensive. Therefore, in addition to the signature-based and behavior-based malware detection techniques, numerous fusion techniques exist that contain the benefits of both [5,6]. The goal of these fusion strategies is to address the shortcomings of signature and behavior-based approaches.

While we work to defend ourselves from malware, cybercriminals continue to create increasingly complex techniques to obtain and steal data and resources. Conventional

methods (i.e., rule-based, graph-based, and entropy-based) for analyzing and detecting malware focus on matching known malicious signatures to alleged malicious programs. Such static solutions require a known harmful signature, rendering them unsatisfactory against new (e.g., zero-day) attacks, and depend on end users to maintain system updates. Attackers are aware that these methods may also be vulnerable to obfuscation, such as code obfuscation to avoid detection against known signatures [7]. Hence, it is necessary to update and build malware detection mechanisms that are capable of withstanding significant attacks [8].

Machine learning offers the potential to construct malware detectors that are capable of combating newer versions of malware, and different supervised and unsupervised-algorithm-based machine learning methods have been reported in the literature [9–11]. More specifically, ensemble learning approaches have been utilized and achieved excellent results in malware detection [12–17]. In most cases, ensemble learning algorithms yield superior results as compared to individual classification algorithms, i.e., support vector machine, decision tree, naive Bayes, and neural networks. However, although classifier ensembles demonstrate a significant performance, the majority of these ensembles are deployed in a restricted manner without adequate hyperparameter tuning. Moreover, the performance of classifier ensembles is validated using a single dataset; consequently, no generalizable results are produced.

The tree-based ensemble technique is an ensemble learning paradigm in which a collection of base learners (e.g., decision trees or CART) are constructed and combined from the training data [18]. For instance, random forest [19] is comprised of a large number of individual decision trees that operate as an ensemble. It uses feature randomness to generate an uncorrelated forest of decision trees. In a similar fashion, the gradient boosting decision tree algorithms combine a collection of individual decision trees to form an ensemble. However, unlike random forest, the decision trees in gradient boosting are constructed serially (e.g., additively). Gradient boosting decision tree algorithms have recently been proposed and have demonstrated remarkable results in many domains, such as protein–protein interaction prediction [20], neutronic calculation [21], human activity recognition [22], etc. However, their performance in classifying and detecting malware remains questionable. This motivated us to employ ensembles of tree-based algorithms to classify PE malware. This paper makes the following contributions to the current literature.

(a) Fine-tuned tree-based classifier ensembles, i.e., random forest [19], XGBoost [23], CatBoost [24], GBM [25], and LightGBM [26], to detect PE malware are employed.
(b) The performance differences between classifier ensembles over the most recent datasets, i.e., BODMAS [27], Kaggle, and CIC-MalMem-2022 [28] are benchmarked using statistical significance tests. This study is among the first to utilize the most recent malware BODMAS and CIC-MalMem-2022 datasets. On the BODMAS and CIC-Malmem-2022 datasets, our proposed approaches outperform other baselines with a 99.96% and 100% accuracy rate, respectively.
(c) An in-depth exploratory analysis of each malware dataset is presented to better understand the characteristics of each malware dataset. The analysis includes a feature correlation analysis and t-SNE visualization of pairs of samples' similarities.

The remainder of the paper is structured as follows. An overview of PE malware detection based on classifier ensembles is provided in Section 2. Next, we present the background of tree-based classifier ensembles and datasets in Section 3. Section 4 discusses the experimental results, and in the end, Section 5 concludes the paper.

2. Related Work

Ucci et al. [7], Maniriho et al. [10] provide the machine learning taxonomy for malware analysis, while [11] present an overview of malware analysis in CPS and IoT. Malware analysis can be accomplished via either static or dynamic analysis, or a mix of the two, depending on how the information extraction procedure is carried out. Approaches based on static analysis evaluate the content of samples without necessitating their execution,

whereas dynamic analysis examines the behavior of samples by executing them. This study analyzes a static analysis of PE files, since it can yield a plethora of useful information, e.g., the compiler and symbols used.

Meanwhile, machine learning techniques were largely employed in malware detection [29,30]. Malware samples were examined and the extracted features are used to train the classification algorithm. An overview of the machine learning techniques used for the classification of malware is provided in the following. We particularly explore malware detectors that employ at least one ensemble learning technique. Vadrevu et al. [31], Mills et al. [17], Uppal et al. [32], Kwon et al. [33] utilized random forest for malware detection based on PE file characteristics and networks. Furthermore, Mao et al. [34], Wüchner et al. [35], Ahmadi et al. [36] developed a random forest classifier to detect malware using various features, such as system calls, file system, and Windows registry. Amer and Zelinka [13] proposed an ensemble learning strategy to address the shortcomings of the existing commercial signature-based techniques. The proposed technique was able to focus on the most salient features of malware PE files by lowering the dimensionality of the data. Dener et al. [37] and Azmee et al. [38] compared the use of various machine learning algorithms to detect PE malware and showed that XGBoost and logistic regression were the best-performing methods.

Liu et al. [39] employed data visualization and adversarial training on ML-based detectors to effectively detect the various types of malware and their variants in order to address the current issues in malware detection, such as the consideration of attacks from adversarial examples and the massive growth in malware variants. In [40], a deep feature extraction technique for malware analysis was addressed in light of the current progress in deep learning. Deep features were obtained from a CNN and were fed to an SVM classifier for malware classification. Moreover, a CNN ensemble for malware classification was proposed in [15,16]. The proposed architecture was constructed in a stacked fashion, with a machine learning algorithm providing the final classification. A meta-classifier was selected after various machine learning algorithms were analyzed and evaluated. Most recently, Hao et al. [41] proposed a CNN-based feature extraction and a channel-attention module to reduce the information loss in the process of feature image generation of malware samples. Specific deep learning architectures, such as a deep belief network and transformer-based classifier, were also considered when classifying Android [42,43] and PE malware [44], respectively. Table 1 presents a summary of the existing malware detectors described in the literature.

Table 1. Summarization of the existing PE malware detectors.

Study	Algorithm(s)	Data Set	Validation Technique	Best Result
Mills et al. [17]	RF	Private	7-CV	-
Vadrevu et al. [31]	RF	Private	CV and Holdout	TPR: 90%, FPR: 0.1%
Uppal et al. [32]	NB, DT, RF, and SVM	Private	10-CV	Accuracy: 98.5%
Kwon et al. [33]	RF	Private	10-CV	TPR: 98.0%, FPR: 2.00%, F1: 98.0%, AUC: 99.8%
Mao et al. [34]	RF	Private	Repeated hold-out	TPR: 99.88%, FPR: 0.1%
Wüchner et al. [35]	RF	Malicia	10-CV	DR: 98.01%, FPR: 0.48%
Ahmadi et al. [36]	XGBoost	Kaggle	5-CV	Accuracy: 98.62%
Amer and Zelinka [13]	RF and extra trees	Kaggle	Hold-out	Accuracy: 99.8%, FPR: 0.2%
Liu et al. [39]	CNN and autoencoder	MS BIG and Ember	10-CV	Accuracy: 96.25%
Asam et al. [40]	CNN and SVM	MalImg	Hold-out	Accuracy: 98.61%, precision: 96.27%, recall: 96.30%, F1: 96.32%
Azeez et al. [15]	1D CNN and Extra trees	Kaggle	10-CV	Accuracy: 100%, precision: 100%, recall: 100%, F1: 100%
Damaševičius et al. [16]	Stacked CNN	ClaMP	10-CV	Accuracy: 99.9%, precision: 99.9%, recall: 99.8%, F1: 99.9%

Table 1. *Cont.*

Study	Algorithm(s)	Data Set	Validation Technique	Best Result
Hou et al. [42]	DBN	Comodo cloud	10-CV	Accuracy; 96.66%
Hou et al. [43]	DBN and SAEs	Comodo cloud	10-CV	Accuracy: 96.66%
Azmee et al. [38]	XGBoost	Kaggle	10-CV	Accuracy: 98.6%, AUC: 0.99, TPR: 99.0%, FPR: 3.7%
Jingwei et al. [41]	CNN	MS BIG and BODMAS	10-CV	(MS BIG) accuracy: 99.40%, (BODMAS) accuracy: 99.26%
Lu et al. [44]	Transformer	BODMAS and MS BIG	Hold-out	(MS BIG) accuracy: 98.17%, F1: 98.14%, (BODMAS) accuracy:96.96%, F1: 96.96%
Dener et al. [37]	Logistic regression	CIC-MalMem-2022	Repeated hold-out	Accuracy: 99.97%

3. Materials and Methods

This study evaluates the performance of ensembles of tree-based classifiers in detecting PE malware. Figure 1 depicts the stages involved in our comparative analysis. Several tree-based ensemble approaches are trained on three distinct PE malware training datasets in order to generate classification models. The performance of classification models is then determined by validating them on a testing dataset. Finally, a two-step statistical significance test is then utilized to evaluate the performance benchmarks. In the following section, we provide a brief summary of the malware datasets and tree-based classifier ensembles utilized in this study.

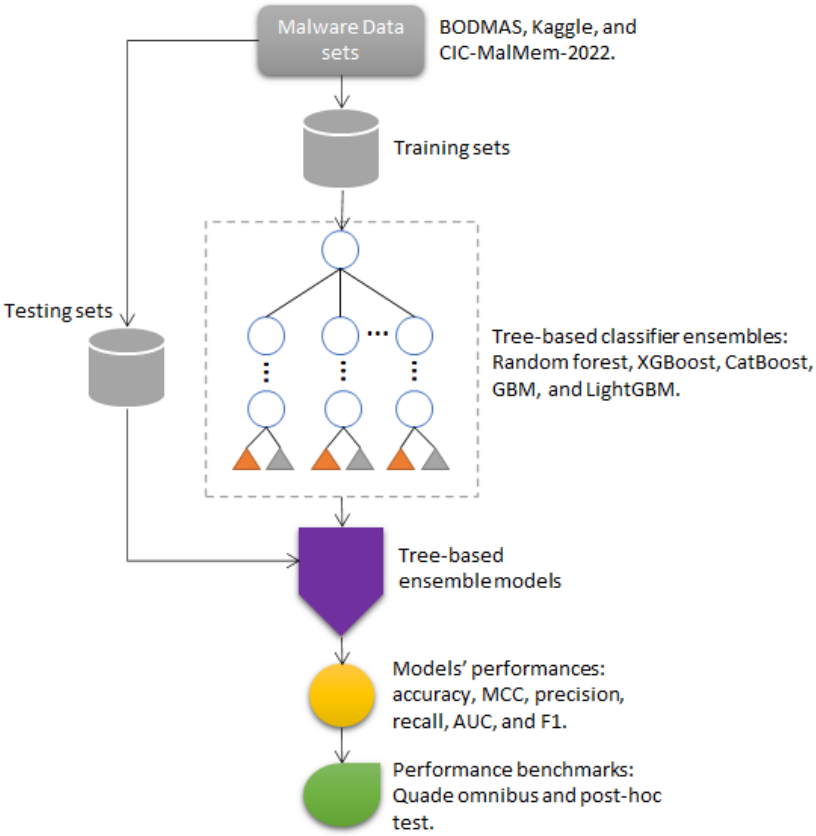


Figure 1. Performance comparison methodology of tree-based ensembles for PE malware detection.

3.1. Datasets

One of the most problematic aspects of using machine learning to solve malware detection problems is producing a realistic feature set from a large variety of unidentified portable executable samples. In essence, the dataset used to train machine learning models determines their level of sophistication. Hence, developing a solid, labeled dataset that represents all analyzed samples is more helpful for malware detection. In light of this, we utilize more recent public datasets that depict the characteristics and attack behaviors of contemporary malware:

(a) BODMAS [27]
The dataset contains 57,293 malicious and 77,142 benign samples (134,435 in total). The malware samples were arbitrarily picked each month from the internal malware database of a security company. The data were collected between 29 August 2019 and 30 September 2020. The benign samples were gathered between 1 January 2007 and 30 September 2020. In order to reflect benign PE binary distribution in real-world traffic, the database of the security company is also processed for benign samples. In addition, SHA-256 hash, the actual PE binary, and a pre-extracted feature vector were given for each malicious sample, whereas only SHA-256 hash and the pre-extracted feature vector were provided for each benign sample. BODMAS is comprised of 2381 input feature vectors and 1 class label feature, of which 0 is labeled as benign and 1 is labeled as malicious.

(b) Kaggle (https://tinyurl.com/22z7u898, access on 25 August 2022)
The dataset was developed using a Python library called *pefile* (https://tinyurl.com/w75zewvr, accessed on 25 August 2022), which is a multi-platform module used to parse and work with PE files. Kaggle dataset contains 14,599 malicious and 5012 benign samples (19,611 in total). The dataset is comprised of 78 input features, denoting PE header files and one class label attribute.

(c) CIC-MalMem-2022 [28]
Unlike the two above-mentioned datasets, CIC-MalMem-2022 is an obfuscated malware dataset that is intended to evaluate memory-based obfuscated malware detection algorithms. The dataset was designed to mimic a realistic scenario as accurately as possible using reowned malware. Obfuscated malware comprises malicious software that conceals itself to escape detection and eradication. The dataset consists of an equal ratio of malicious and benign memory dumps (58,596 samples in total). In addition, CIC-MalMem-2022 is made up of 56 features that serve as inputs for machine learning algorithms.

3.2. Tree-Based Ensemble Learning

The tree-based ensemble is a non-ordinary learning paradigm that constructs and combines a set of base learners (e.g., decision trees or CART) as opposed to the commonplace learning paradigm that attempts to construct individual learners from training data. Normally, an ensemble is formed in two processes, i.e., by first producing the base learners and then integrating them. For a decent ensemble, it is commonly considered that the base learners must be as accurate and diversified as possible [18]. This study considers four tree-based ensemble learning algorithms. It is worth mentioning that tuning the hyperparamters for each algorithm is carried out using *random search* approach [45].

(a) Random forest [19]
As its name implies, a random forest is a tree-based ensemble in which each tree is dependent on a set of random variables. The original formulation of random forest algorithm provided by Breiman [19] is as follows. A random forest employs trees $h_j(\mathcal{X}, \Omega)$ as its base learners. For training data $\mathscr{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\alpha, y_\alpha)\}$, where $\mathbf{x}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,p})^T$ represents the p predictors and y_i represents the response, and a specific manifestation ω_j of Ω_j, the fitted tree is given as $\hat{h}_j(\mathbf{x}, \omega_j, \mathscr{D})$. More precisely, the steps involved in the random forest algorithm are described in Algorithm 1.

We use a fast random forest implementation called *ranger* [46], available in *R*, which is suitable for high-dimensional data such as ours. The list of random forest's hyperparameters for each malware dataset is provided in Table 2. We set the search space for each hyperparameter tuning is as follows. Number of trees = {50, 100, 250, 500, 750, 1000}, split rule = {'gini','extratrees'}, minimum node size = {1, 2, ..., 10}, *mtry* = number of features × {0.05, 0.15, 0.25, 0.333, 0.4}, sample fraction = {0.5, 0.63, 0.8}, and *replace* = {TRUE, FALSE}.

Algorithm 1: A common procedure of random forest algorithm for classification task.

Training:
Require: Original training set $\mathscr{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_\alpha, y_\alpha)\}$, with $\mathbf{x}_i = (\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,p})^T$
1. **for** $j = 1$ to J
2. Perform a bootstrap sample $\mathscr{D}_j$ of size α from $\mathscr{D}$.
3. Using binary recursive partitioning, fit a tree on $\mathscr{D}_j$.
4. **end for**

Testing:
Require: An instance to be classified $\mathbf{x}$.
1. $\hat{f}(\mathbf{x}) = \arg\max_y \sum_{j=1}^{J} I(\hat{h}_j(\mathbf{x}) = y)$

where $\hat{h}_j(\mathbf{x})$ denotes the response variable at $\mathbf{x}$ using the j-th tree.

Table 2. The final learning parameters of random forest used for each dataset after performing a *random search*.

Hyperparameter	BODMAS	Kaggle	CIC-MalMem-2022
Number of trees	100	1000	500
Split rule	'gini'	'gini'	'extratrees'
Minimum node size	4	8	6
mtry	119	30	18
Sample fraction	0.63	0.80	0.63
replace	FALSE	FALSE	TRUE

(b) Gradient Boosting Decision Trees

In this paper, we also considered various tree-based boosting ensemble approaches for malware detection, such as XGBoost [23], CatBoost [24], GBM [25], and LightGBM [26]. As a rule, GBDT ensembles are a linear additive model, where a tree-based classifier (e.g., CART) was utilized as their base model. Let $\mathscr{D} = \{(\mathbf{x}_i, y_i) | i \in \{1, \ldots, \alpha\}, \mathbf{x}_i \in \mathbb{R}^\eta, y_i \in \mathbb{R}\}$ denote the malware dataset comprising η features and α samples. Considering a collection of j trees, the prediction output $y(\hat{\mathbf{x}})^j$ for an input $\mathbf{x}$ is obtained by calculating the predictions from each tree $y(\hat{\mathbf{x}})^j$, as shown in the following formula.

$$y(\hat{\mathbf{x}})^j = \sum_{i=1}^{j} f_i(\mathbf{x}) \tag{1}$$

where f_i represents the output of the i-th regression tree of the j-tree ensemble. GBDTs minimize a regularized objective function Obj^t in order to create the $(j+1)$-th tree, as follows.

$$\min\{Obj(f)^t\} = \min\{\Omega(f)^t + \Theta(f)^t\} \tag{2}$$

where $\Omega(f)^t$ represents loss function and $\Theta(f)^t$ is a regularization function to control over-fitting. The loss function $\Omega(f)^t$ measures the difference between the prediction $\hat{y}_i$ and the target y_i. On the other hand, the regularization function is defined as $\Theta(f)^t = \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2$, where T and w indicate the number of leaves and leaf weights in the tree, respectively.

(i) XGBoost [23]
XGBoost is a scalable end-to-end tree-boosting strategy that generates a large number of sequentially trained trees. Each succeeding tree corrects the errors made by the preceding one, resulting in an efficient classification model. Through sparsity-aware metrics and multi-threading approaches, XGBoost not only addresses the algorithm's overfitting problem, but also boosts the speed of most real-world computational tasks. This study utilizes two different XGBoost implementations, such as native implementation in *R* [47] and *H2O* [48]. We set the search space of native XGBoost's hyperparameters are as follows. Maximum depth = {2, 3, ..., 24}, eta = {0, 0.1, 0.2, ..., 1.0}, subsample = {0.5, 0.6, 0.7, 0.8}, and column sample by tree = {0.5, 0.6, 0.7, 0.8, 0.9}. Moreover, we set the search space of XGBoost's hyperparameters implemented in *H2O* are as follows. Maximum depth = {1, 3, 5, ..., 29}, sample rate = {0.2, 0.3, ..., 1}, column sample rate = {0.2, 0.21, 0.22, ..., 1}, column sample rate per tree = {0.2, 0.21, 0.22, ...,1}, and minimum rows = {0, 1, ..., $\log_2 \times$ number of rows-1}. The final learning parameters for both XGBoost implementations are presented in Table 3.

Table 3. The final learning parameters of XGBoost used for each dataset after performing a *random search*.

Hyperparameter	BODMAS	Kaggle	CIC-MalMem-2022
Native			
Maximum depth	11	19	19
eta	0.2	0.3	0.1
Subsample	0.6	0.8	0.6
Column sample by tree	0.7	0.5	0.6
H2O			
Maximum depth	24	23	26
Sample rate	0.52	0.94	0.99
Column sample rate	0.42	0.62	0.6
Column sample rate per tree	0.6	0.25	0.5
Minimum rows	2	2	2

(ii) CatBoost [24]
CatBoost is built with symmetric decision trees. It is acknowledged as a classification algorithm that is capable of producing an excellent performance and ten times the prediction speed of methods that do not employ symmetric decision trees. CatBoost, unlike other GBDT algorithms, is able to accommodate gradient bias and prediction shift to increase the accuracy of predictions and generalization ability of large datasets. In addition, CatBoost is comprised of two essential algorithms: ordered boosting, which estimates leaf values during tree structure selection to avoid overfitting, and a unique technique for handling categorical data throughout the training process. An implementation of CatBoost in *R* is employed in this paper, whereas the search space of each hyperparameter is considered as follows. Depth = {1, 2, ..., 10}, learning rate = {0.03, 0.001, 0.01, 0.1, 0.2, 0.3}, l2 leaf regularization = {3, 1, 5, 10, 100}, border count = {32, 5, 10, 20, 50, 100, 200}, and boosting type = {"Ordered", "Plain"}. The final learning parameters of CatBoost for each malware dataset are given in Table 4.

Table 4. The final learning parameters of CatBoost used for each dataset after performing a *random search*.

Hyperparameter	BODMAS	Kaggle	CIC-MalMem-2022
Depth	10	4	2
Learning rate	0.2	0.2	0.2
L2 leaf regularization	5	3	5
Border count	100	100	50
Boosting type	"Plain"	"Plain"	"Ordered"

(iii) Gradient boosting machine [25]
GBM is the first implementation of GBDT to utilize a forward learning technique. Trees are generated in a sequential manner, with future trees being dependent on the results of the preceding trees. Formally, GBM is achieved by iteratively constructing a collection of functions $f^0, f^1, \ldots, f^t$, given a loss function $\Omega(y_i, f^t)$. We can optimize our estimates of y_i by discovering another function $f^{t+1} = f^t + h^{t+1}(\mathbf{x})$, such that h^{t+1} reduces the estimated value of the loss function. In this study, we adopt GBM implementation in H2O, whereas the hyperparameters' search space is specified as follows. Maximum depth = {1, 3, 5, ..., 29}, sample rate = {0.2, 0.3, ..., 1}, column sample rate per tree = {0.2, 0.21, 0.22, ..., 1}, column sample rate change per level = {0.9, 0.91, ..., 1.1}, number of bins = $2^{\{4,5,\ldots,10\}}$, and minimum rows = {0, 1, ..., $\log_2 \times$ number of rows $-$ 1}. Table 5 shows a list of all the final GBM hyperparameters that were used on each malware dataset.

Table 5. The final learning parameters of GBM used for each dataset after performing *random search*.

Hyperparameter	BODMAS	Kaggle	CIC-MalMem-2022
Maximum depth	24	25	27
Sample rate	0.52	0.44	0.72
Column sample rate per tree	0.42	0.64	0.61
Column sample rate change per level	1.02	1.04	0.92
Number of bins	64	512	1024
Minimum rows	2	2	8

(iv) LightGBM [26]
LightGBM is an inexpensive gradient boosting tree implementations that employs histogram and leaf-wise techniques to increase both processing power and prediction precision. The histogram method is used to combine features that are incompatible with each another. Before generating a n-width histogram, the core idea is to discretize continuous features into n integers. Based on the discretized values of the histogram, the training data are scanned to locate the decision tree. The histogram method considerably reduces the runtime complexity. In addition, in LightGBM, the leaf with the greatest splitting gain was found and then divided using a leaf-by-leaf strategy. Leaf-wise optimization may result in overfitting and a deeper decision tree. To ensure great efficiency and prevent overfitting, LightGBM includes a maximum depth constraint to leaf-wise. In this study, we employed a LightGBM implementation in *R* with the following hyperparameter search space; Maximum bin = {100, 255}, maximum depth = {1, 2, ..., 15}, number of leaves = $2^{(\{1,2,\ldots,15\}}$, minimum data in leaf = {100, 200, ..., 1000}, learning rate = {0.01, 0.3, 0.01}, lambda l1 = {0, 10, 20, ..., 100}, lambda l2 = {0, 10, 20, ..., 100}, feature fraction = {0.5, 0.9}, bagging fraction = {0.5, 0.9}, path smooth = {1×10^{-8}, 1×10^{-3}}, and minimum gain to split = {0, 1, 2, ..., 15}.

Table 6 contains the list of all final LightGBM hyperparameters used for each malware dataset.

Table 6. The final learning parameters of LightGBM used for each dataset after performing *random search*.

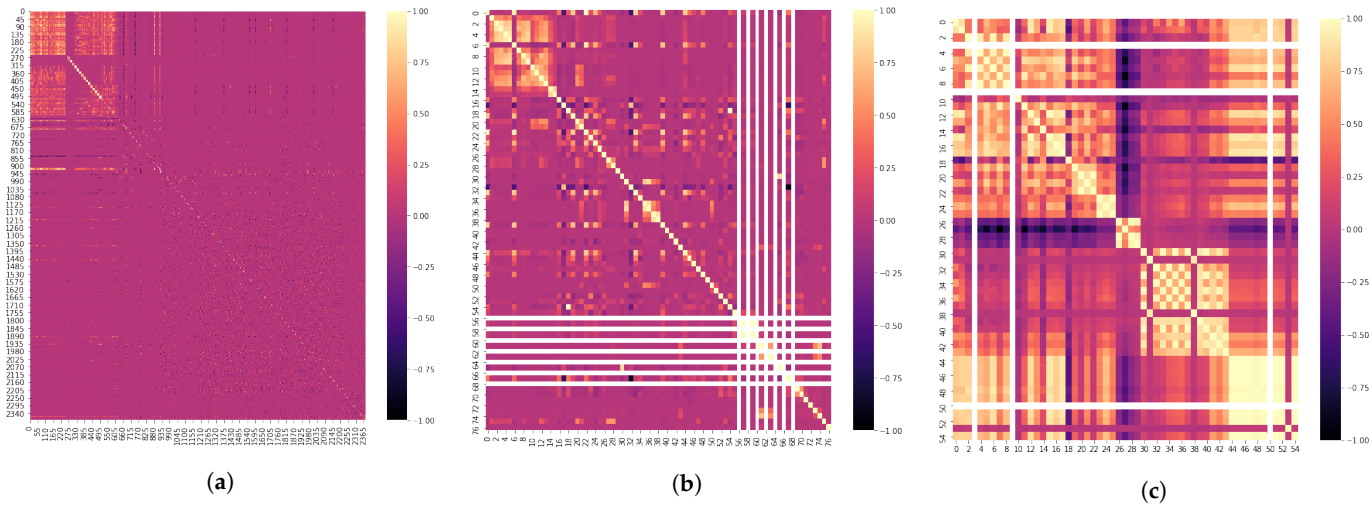
Hyperparameter	BODMAS	Kaggle	CIC-MalMem-2022
Maximum bin	100	100	255
Maximum depth	10	9	3
Number of leaves	8192	8	512
Minimum data in leaf	1000	800	700
Learning rate	0.29	0.27	0.07
Lambda l1	40	0	0
Lambda l2	90	20	90
Feature fraction	0.5	0.9	0.9
Bagging fraction	0.5	0.5	0.5
Path smooth	0.001	1×10^{-8}	0.001
Minimum gain to split	2	15	11

4. Result and Discussion

This section analyzes and discusses the results of the tree-based classifier ensembles applied to malware classification. The results of exploratory analysis are presented first, followed by a performance comparison between the tree-based ensemble models.

4.1. Exploratory Analysis

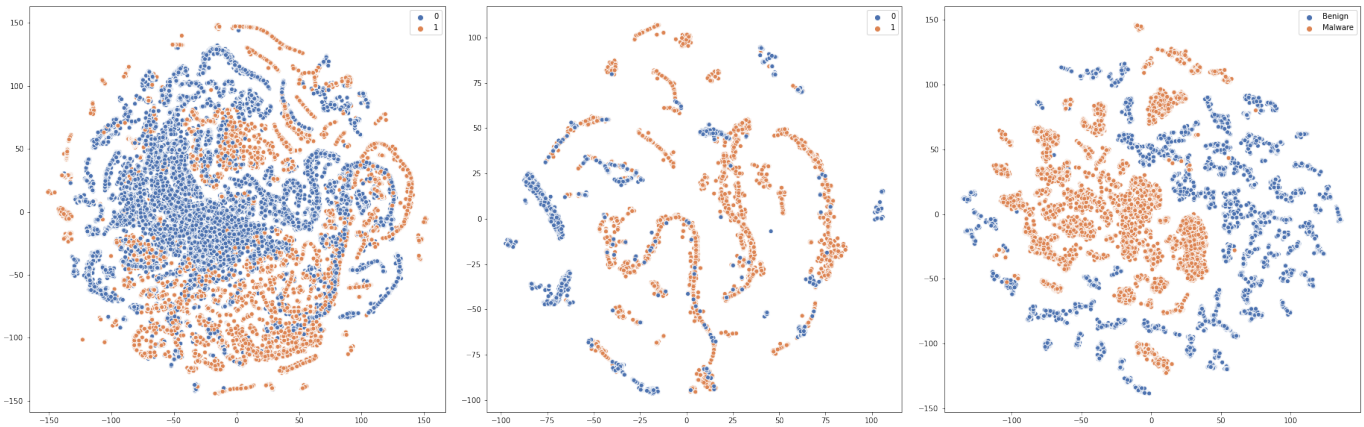
We first provide a correlation analysis between multiple variables in each malware dataset. Figure 2 shows the correlation coefficient score matrix measured by Pearson correlation. Correlation analysis is useful to understand the relationship between variables in a dataset, since the Good input features of a dataset should have a high correlation with target features, but should be uncorrelated with each other. Figure 2 confirms that both BODMAS and Kaggle datasets have fewer uncorrelated features than CIC-MalMem-2022. Hence, to mitigate the curse of dimesionality, it is strongly recommended to employ feature selection before employing a machine learning method on CIC-MalMem-2022. Highly correlated features have a negligible effect on the output prediction but raise the computational cost.



(**a**) (**b**) (**c**)

Figure 2. Feature correlation analysis of each malware dataset: (**a**) BODMAS, (**b**) Kaggle, and (**c**) CIC-MalMem-2022.

In addition, we ran a t-SNE algorithm [49] with a learning rate = 5000 and perplexity = 100. The t-SNE is an approach that converts a set of high-dimensional points to two dimensions in such a way that, ideally, close neighbors remain close and far points remain far. Figure 3 provides a spatial representation of the dataset in two dimensions. The t-SNE provides a pliable border between the local and global data structures. It also estimates the size of each datapoint's local neighborhood based on the local density of the data by requiring each conditional probability distribution to have the same perplexity (e.g., Gaussian kernel). Furthermore, Figure 3 demonstrates that both BODMAS and Kaggle datasets are highly imbalanced as compared with CIC-MalMem-2022.



(**a**) (**b**) (**c**)

Figure 3. Two-dimensional visualization of instance pairs using t-SNE technique of each malware dataset: (**a**) BODMAS, (**b**) Kaggle, and (**c**) CIC-MalMem-2022.

4.2. Comparison Analysis

In the experiment, we employed a k cross-validation technique ($k = 10$), where the final performance outcome for each tree-ensemble model is the mean of the ten folds. The performance of each model was measured based on six performance metrics, such as accuracy, MCC, precision, recall, AUC, and F1. These metrics are chosen to provide more accurate estimates of the behavior of the classifier ensembles under the experiment. Especially, Chicco et al. [50] have shown that MCC is more informative than accuracy and F1, which yield reliable estimates when used to balanced datasets, but misleading outcomes when applied to imbalanced data sets. For a binary classification problem, the outcome of a tree-based classifier ensemble is typically derived from a contigency matrix, $\mathcal{T} = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$, where TP is true positive, FN is false negative, FP is false positive, and TN is true negative. Let $\xi_+ = TP + FN$ and $\xi_- = TN + FP$ be the number of samples labeled as malware and non-malware, respectively. Hence, the performance metrics used in this study can be calculated as follows.

$$\text{Accuracy} = \frac{TP + TN}{\xi_+ + \xi_-} \tag{3}$$

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{((TP + FP) \times \xi_- \times (TN + FN) \times \xi_+)^{1/2}} \tag{4}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\text{Recall} = \frac{TP}{\xi_+} \tag{6}$$

$$\text{AUC} = \int_0^1 \text{Recall} \times \frac{FP}{\xi_-} d\frac{FP}{\xi_-} = \int_0^1 \text{Recall} \times \left(\frac{FP}{\xi_-}\right)^{-1}(x)dx \tag{7}$$

$$\text{F1} = \frac{2TP}{2TP + FP + FN} \tag{8}$$

Figure 4 presents the performance score of each algorithm on each dataset. Overall, considering MCC as a performance indicator, LightGBM is the worst-performing algorithm, while XGBoost (native) is the best-performing on BODMAS and Kaggle datasets, followed by GBM (H2O). Interestingly, random forest has also performed well on the remaining dataset. Using accuracy as a performance metric, it is also apparent that there are modest performance disparities amongst algorithms (e.g., all algorithms achieve 100% accuracy). Consequently, our results support the findings stated by [50]. In Table 7, we provide the performance average of each algorithm over various datasets and demonstrate that XGBoost (native) is superior to any competitors on the board in terms of accuracy, MCC, and precision metrics. On the other hand, when recall, AUC, and F1 metrics are utilized, GBM (H2O) shows a superior performance.

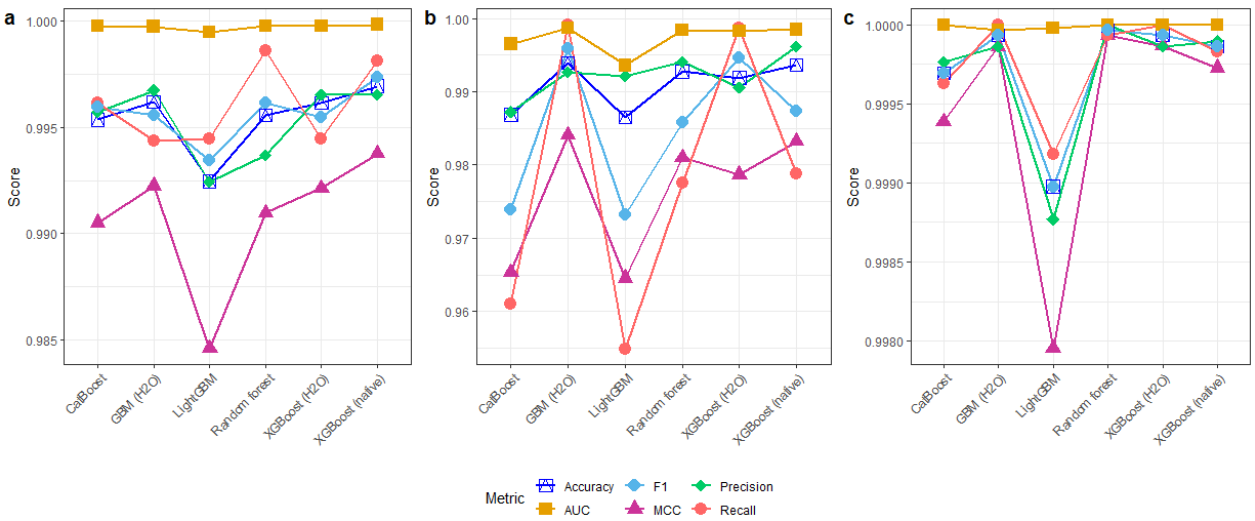


Figure 4. Performance comparison of various tree-based ensemble models on different datasets, i.e., (**a**) BODMAS, (**b**) Kaggle, and (**c**) CIC-MalMem-2022.

Table 7. Performance average of each ensemble technique over various malware datasets.

Ensemble Algorithms	Accuracy	MCC	Precision	Recall	AUC	F1
CatBoost	0.9940	0.9851	0.9943	0.9856	0.9988	0.9899
XGBoost (native)	0.9968	0.9922	0.9975	0.9923	0.9994	0.9949
LightGBM	0.9927	0.9823	0.9945	0.9828	0.9977	0.9885
Random forest	0.9961	0.9906	0.9959	0.9921	0.9994	0.9940
GBM (H2O)	0.9967	0.9920	0.9964	0.9978	0.9995	0.9971
XGBoost (H2O)	0.9960	0.9902	0.9956	0.9977	0.9994	0.9967

This section includes a two-step statistical significance test using Quade omnibus test and Quade post-hoc test [51] to better comprehend the performance difference between tree-based ensemble models. Using a significant threshold $\alpha = 0.05$ and MCC as a performance indicator, the Quade omnibus test demonstrates that at least one classifier performs differently than others (p-value = 0.01725). Since we found significance in the previous test, we then applied the Quade post-hoc test to determine the pairwise performance difference between classifiers. Here, we considered XGBoost (native) as a control classifier for comparison with the remaining algorithms. Table 8 depicts the p-value of the pairwise

comparison. It is clear that the performance differences between XGBoost and the other algorithms are not statistically significant (p-value > 0.05).

Table 8. The p-value of Quade post-hoc, in which XGBoost (native) is used as a control algorithm.

CatBoost	XGBoost (Native)	LightGBM	Random Forest	GBM (H2O)	GBM (H2O)
0.250153	-	0.125201	0.7014781	0.6092802	0.8983268

To demonstrate the efficacy of tree-based ensemble models for malware detection, we compared our performance findings to those of previous studies for each dataset. Table 9 denotes the performance comparisons in terms of several performance measures, such as accuracy, precision, recall, and F1. Please note that the comparison is conducted as objectively as possible, given that the prior experiment may have been conducted under different settings, such as validation techniques and the number of training and testing samples. Nevertheless, this study shows that the top-performing tree-based ensemble examined for each dataset outperforms prior research, with a comparable result. More precisely, GBM (H2O), XGBoost (native), and random forest are the best performers on the Kaggle, BODMAS, and CIC-MalMem-2022 datasets, respectively, which also outperform other state-of-the-art malware detection techniques available in the recent literature.

Table 9. Performance comparisons over existing studies. The best performance value on each dataset is shown in bold.

Study	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Kaggle				
Hou et al. [42]	93.68	93.96	93.36	93.68
Hou et al. [43]	96.66	96.55	96.76	96.66
Azmee et al. [38]	98.60	96.30	99.00	-
This study (GBM (H2O))	**99.39**	**99.27**	**99.92**	**99.59**
BODMAS				
Jingwei et al. [41]	99.29	98.07	98.26	94.23
Lu et al. [44]	96.96	-	-	96.96
This study (XGBoost (native))	**99.96**	**99.65**	**99.81**	**99.73**
CIC-MalMem-2022				
Dener et al. [37]	99.97	99.98	99.97	99.97
This study (Random forest)	**100.00**	**100.00**	99.99	**100.00**

5. Conclusions

This article examined tree-based ensemble learning algorithms that analyze PE malware. Several tree-based ensemble techniques, including random forest, XGBoost, CatBoost, GBM, and LightGBM, were assessed based on a number of performance criteria, such as accuracy, MCC, precision, recall, AUC, and F1. In addition, we incorporated cutting-edge malware datasets to comprehend the most recent attack trends. This work contributed to the prior research in several ways, including by providing a statistical comparison of fine-tuned tree-based ensemble models utilizing several malware datasets. Furthermore, this article can be expanded in a number of ways, including by looking at the explainability of tree-based ensemble models and signature-based malware classification. Furthermore, a deep neural network model for tabular data, such as TabNet [52], has been underexplored in this application domain, providing a new direction for future research. Finally, it is anticipated that tree-based PE malware detection will be deployed in various real-world settings, such as in host, network, and cloud-based malware detection components.

Author Contributions: Conceptualization, M.H.L.L. and B.A.T.; methodology, B.A.T.; validation, M.H.L.L.; investigation, M.H.L.L.; writing—original draft preparation, M.H.L.L.; writing—review and editing, M.H.L.L. and B.A.T.; visualization, B.A.T.; supervision, B.A.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

List of Acronyms

AUC	Area Under ROC Curve.
CART	Classification and Regression Tree.
CNN	Convolutional Neural Network.
CPS	Cyber-Physical Systems.
CV	Cross Validation.
DBN	Deep Belief Network.
DR	Detection Rate.
DT	Decision Tree.
FPR	False Positive Rate.
GBDT	Gradient Boosting Decision Tree.
GBM	Gradient Boosting Machine.
IoT	Internet of Things.
MCC	Matthews Correlation Coefficient.
NB	Naive Bayes.
PE	Portable Executable.
RF	Random Forest.
SAEs	Stacked AutoEncoders.
SVM	Support Vector Machine.
t-SNE	t-Stochastic Neighbor Embedding.
TPR	True Positive Rate.

References

1. Kleidermacher, D.; Kleidermacher, M. *Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development*; Elsevier: Amsterdam, The Netherlands, 2012.
2. Xhafa, F. *Autonomous and Connected Heavy Vehicle Technology*; Academic Press: Cambridge, MA, USA, 2022.
3. Smith, D.J.; Simpson, K.G. *Safety Critical Systems Handbook*; Elsevier: Amsterdam, The Netherlands, 2010.
4. Damodaran, A.; Troia, F.D.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 1–12. [CrossRef]
5. Mahdavifar, S.; Ghorbani, A.A. Application of deep learning to cybersecurity: A survey. *Neurocomputing* **2019**, *347*, 149–176. [CrossRef]
6. Zhang, W.; Wang, H.; He, H.; Liu, P. DAMBA: Detecting android malware by ORGB analysis. *IEEE Trans. Reliab.* **2020**, *69*, 55–69. [CrossRef]
7. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
8. Milosevic, J.; Malek, M.; Ferrante, A. Time, accuracy and power consumption tradeoff in mobile malware detection systems. *Comput. Secur.* **2019**, *82*, 314–328. [CrossRef]
9. Zhang, H.; Xiao, X.; Mercaldo, F.; Ni, S.; Martinelli, F.; Sangaiah, A.K. Classification of ransomware families with machine learning based onN-gram of opcodes. *Future Gener. Comput. Syst.* **2019**, *90*, 211–221. [CrossRef]
10. Maniriho, P.; Mahmood, A.N.; Chowdhury, M.J.M. A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges. *Future Gener. Comput. Syst.* **2022**, *130*, 1–18. [CrossRef]

11. Montes, F.; Bermejo, J.; Sanchez, L.; Bermejo, J.; Sicilia, J.A. Detecting Malware in Cyberphysical Systems Using Machine Learning: A Survey. *KSII Trans. Internet Inf. Syst. (TIIS)* **2021**, *15*, 1119–1139.
12. Singh, J.; Singh, J. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Inf. Softw. Technol.* **2020**, *121*, 106273. [CrossRef]
13. Amer, E.; Zelinka, I. An ensemble-based malware detection model using minimum feature set. *Mendel* **2019**, *25*, 1–10.
14. Atluri, V. Malware Classification of Portable Executables using Tree-Based Ensemble Machine Learning. In Proceedings of the 2019 SoutheastCon, Huntsville, AL, USA, 11–14 April 2019; pp. 1–6. [CrossRef]
15. Azeez, N.A.; Odufuwa, O.E.; Misra, S.; Oluranti, J.; Damaševičius, R. Windows PE Malware Detection Using Ensemble Learning. *Informatics* **2021**, *8*, 10. [CrossRef]
16. Damaševičius, R.; Venčkauskas, A.; Toldinas, J.; Grigaliūnas, Š. Ensemble-Based Classification Using Neural Networks and Machine Learning Models for Windows PE Malware Detection. *Electronics* **2021**, *10*, 485. [CrossRef]
17. Mills, A.; Spyridopoulos, T.; Legg, P. Efficient and Interpretable Real-Time Malware Detection Using Random-Forest. In Proceedings of the International Conference on Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA), Oxford, UK, 3–4 June 2019; pp. 1–8. [CrossRef]
18. Zhou, Z.H. *Ensemble Methods: Foundations and Algorithms*; CRC Press: Boca Raton, FL, USA, 2012.
19. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
20. Chen, C.; Zhang, Q.; Ma, Q.; Yu, B. LightGBM-PPI: Predicting protein-protein interactions through LightGBM with multi-information fusion. *Chemom. Intell. Lab. Syst.* **2019**, *191*, 54–64. [CrossRef]
21. Cai, J.; Li, X.; Tan, Z.; Peng, S. An assembly-level neutronic calculation method based on LightGBM algorithm. *Ann. Nucl. Energy* **2021**, *150*, 107871. [CrossRef]
22. Csizmadia, G.; Liszkai-Peres, K.; Ferdinandy, B.; Miklósi, Á.; Konok, V. Human activity recognition of children with wearable devices using LightGBM machine learning. *Sci. Rep.* **2022**, *12*, 5472. [CrossRef]
23. Chen, T.; He, T.; Benesty, M.; Khotilovich, V.; Tang, Y.; Cho, H.; Chen, K. Xgboost: Extreme gradient boosting. In *R Package Version 0.4–2*; R Foundation for Statistical Computing: Vienna, Austria, 2015; Volume 1, pp. 1–4.
24. Dorogush, A.V.; Ershov, V.; Gulin, A. CatBoost: gradient boosting with categorical features support. *arXiv* **2018**, arXiv:1810.11363.
25. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
26. Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.Y. Lightgbm: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–9.
27. Yang, L.; Ciptadi, A.; Laziuk, I.; Ahmadzadeh, A.; Wang, G. BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In Proceedings of the IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 27 May 2021; pp. 78–84. [CrossRef]
28. Carrier, T.; Victor, P.; Tekeoglu, A.; Lashkari, A.H. Detecting Obfuscated Malware using Memory Feature Engineering. In Proceedings of the ICISSP, Online, 9–11 February 2022; pp. 177–188.
29. Singh, J.; Singh, J. A survey on machine learning-based malware detection in executable files. *J. Syst. Archit.* **2021**, *112*, 101861. [CrossRef]
30. Albishry, N.; AlGhamdi, R.; Almalawi, A.; Khan, A.I.; Kshirsagar, P.R. An Attribute Extraction for Automated Malware Attack Classification and Detection Using Soft Computing Techniques. *Comput. Intell. Neurosci.* **2022**, *2022*, 5061059. [CrossRef]
31. Vadrevu, P.; Rahbarinia, B.; Perdisci, R.; Li, K.; Antonakakis, M. Measuring and detecting malware downloads in live network traffic. In Proceedings of the European Symposium on Research in Computer Security, Egham, UK, 9–13 September 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 556–573.
32. Uppal, D.; Sinha, R.; Mehra, V.; Jain, V. Malware detection and classification based on extraction of API sequences. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 34–27 September 2014; pp. 2337–2342.
33. Kwon, B.J.; Mondal, J.; Jang, J.; Bilge, L.; Dumitraş, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; pp. 1118–1129.
34. Mao, W.; Cai, Z.; Towsley, D.; Guan, X. Probabilistic inference on integrity for access behavior based malware detection. In Proceedings of the International Symposium on Recent Advances in Intrusion Detection, Tokyo, Japan, 2–4 November 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 155–176.
35. Wüchner, T.; Ochoa, M.; Pretschner, A. Robust and effective malware detection through quantitative data flow graph metrics. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Milan, Italy, 9–10 July 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 98–118.
36. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 1–9 March 2016; pp. 183–194.
37. Dener, M.; Ok, G.; Orman, A. Malware Detection Using Memory Analysis Data in Big Data Environment. *Appl. Sci.* **2022**, *12*, 8604. [CrossRef]
38. Azmee, A.; Choudhury, P.P.; Alam, M.A.; Dutta, O.; Hossai, M.I. Performance Analysis of Machine Learning Classifiers for Detecting PE Malware. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 510–517. [CrossRef]

39. Liu, X.; Lin, Y.; Li, H.; Zhang, J. A novel method for malware detection on ML-based visualization technique. *Comput. Secur.* **2020**, *89*, 101682. [CrossRef]
40. Asam, M.; Hussain, S.J.; Mohatram, M.; Khan, S.H.; Jamal, T.; Zafar, A.; Khan, A.; Ali, M.U.; Zahoora, U. Detection of Exceptional Malware Variants Using Deep Boosted Feature Spaces and Machine Learning. *Appl. Sci.* **2021**, *11*, 10464. [CrossRef]
41. Hao, J.; Luo, S.; Pan, L. EII-MBS: Malware Family Classification via Enhanced Instruction-level Behavior Semantic Learning. *Comput. Secur.* **2022**, *112*, 102905.
42. Hou, S.; Saas, A.; Ye, Y.; Chen, L. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In Proceedings of the International Conference on Web-Age Information Management, Nanchang, China, 3–5 June 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 54–66.
43. Hou, S.; Saas, A.; Chen, L.; Ye, Y.; Bourlai, T. Deep neural networks for automatic android malware detection. In Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Sydney, Australia, 31 July–3 August 2017; pp. 803–810.
44. Lu, Q.; Zhang, H.; Kinawi, H.; Niu, D. Self-Attentive Models for Real-Time Malware Classification. *IEEE Access* **2022**, *10*, 95970–95985. [CrossRef]
45. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
46. Wright, M.N.; Ziegler, A. Ranger: A fast implementation of random forests for high dimensional data in C++ and R. *arXiv* **2015**, arXiv:1508.04409.
47. Chen, T.; He, T.; Benesty, M.; Khotilovich, V. Package 'xgboost'. *R Version* **2019**, *90*, 1–66.
48. Cook, D. *Practical Machine Learning with H2O: Powerful, Scalable Techniques for Deep Learning and AI*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2016.
49. Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9*, 2579–2605.
50. Chicco, D.; Tötsch, N.; Jurman, G. The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioBata Min.* **2021**, *14*, 13. [CrossRef] [PubMed]
51. Conover, W.J. *Practical Nonparametric Statistics*; John Wiley & Sons: Hoboken, NJ, USA, 1999; Volume 350.
52. Arik, S.Ö.; Pfister, T. Tabnet: Attentive interpretable tabular learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Virsual Conference, 2–9 February 2021; Volume 35, pp. 6679–6687.