Malicious Malware Identification in Mac OS X using Supervised Learning Techniques

Imran Khan[1], Fazal Wahab[2,*], and Sarath Bodapathi[3]

[1]Department of Informatics, University of Sussex, UK
[2]National University of Computer and Emerging Sciences, Faisalabad, Pakistan
[3]Department of Engineering and Computer Science, University of Hertfordshire, UK
Email: imran.khan@sussex.ac.uk (I.K.), studentpk2024@gmail.com (F.W.), sarath.bodapathi@herts.ac.uk (S.B.)
[*]Corresponding author
Manuscript received August 6, 2025; accepted October 11, 2025; published December 25, 2025

***Abstract*—The proliferation of malicious software is rapidly escalating, posing significant challenges in the identification and classification of emerging threats. The inefficiency of conventional approaches employed in malware detection has led to the exploration of Machine Learning (ML) algorithms as a promising solution. This study attempts to enhance existing cybersecurity techniques and improve system protection against potential threats. The objective is to create an intelligent malware detection system specifically designed for accurately and consistently identifying hazardous malware in the Macintosh Operating System X (Mac OS X). We determine the optimal supervised model's hyperparameter value to improve Mac OS X's security. We evaluate the efficacy of these supervised algorithms in comparison to boosting approaches. The experimental findings demonstrate that both the histogram-based boosting approaches and the random forest classifier have exhibited remarkable efficacy in identifying malware on Mac OS X. The algorithms achieved a validation data accuracy of 94% and 91%, and a testing data accuracy of 86% and 89%, respectively. The results are also compared with other state-of-the-art methods. These results are crucial for Mac OS X users, as they ensure the protection of their systems and data against potential threats. Additionally, this approach can also be implemented with different systems for security purposes.**

***Keywords*—intelligent system, malware identification, artificial intelligence, supervised learning, machine learning**

I. Introduction

In today's modern world, computers are an integral part of everyone's everyday lives. It is a significant understatement to argue that modern humans exist in a technologically networked environment. Computers have become increasingly essential in various aspects of modern life, including education, banking, purchasing home items, and recreation. Computers are valuable for organizations, serving not only to carry out and document business transactions but also to facilitate other company operations, including inventory control, payroll management, and more [1]. Furthermore, the Internet has produced a globally interconnected digital domain that enables communication, the exchange of information, commercial endeavors, event coordination, live broadcasting, and social assistance throughout the globe. However, the same technologies and infrastructure that enable positive relationships and creativity are also exploited by criminals to harm and intimidate people, companies, and official organizations [2]. Malware is a computer program specifically created to intentionally inflict damage against computer systems, servers, networks, mobile devices and other autonomous systems. Malicious software presents substantial threats to computer security and undermines the confidentiality of users. It can infiltrate secure systems without authorization, resulting in system unavailability and the unauthorized acquisition of confidential information. Besides, malware is most often used with the aim of gaining some financial advantages, taking part in extortion, and other various criminal activities [3, 4]. Malware has proven to be a serious and persistent risk in the cyber environment and considerable measures and practices should be established to identify, thwart, and minimize its effects. Appropriate solutions are required to protect systems and information against the negative impact of malicious software [5]. The risk of malicious software to the security and privacy of Macintosh Operating System (Mac OS) is an existing issue. Due to the ever-increasing popularity of Mac OS, it has recently become a major target of malware of different types, including spyware, ransomware, adware, and Trojans. An example of such is ransomware that encrypts user data and requires payment to unlock it, with risks of losing data and suffering financial damage [6]. Adware is annoying to the user with its pop-up advertisements and may slow down the system. Spyware is a program that functions in the background without the knowledge of the user and gathers sensitive data, thus it attacks privacy. Trojan horses masquerade as legit software and offer backdoor access to systems, allowing attackers to perform malicious actions [2]. Proactive steps are needed in order to counter these risks. It is possible to use reliable antivirus software designed specifically for Mac OS to check and possibly block malware infection. Users would be advised to be wary when using the internet, downloading of files or opening of email attachments, especially those that are not trusted. It is also important to regularly update the operating system and the applications with the latest security patches to tackle the vulnerabilities and harden the system defenses. Moreover, it is possible to reduce the effectiveness of the possible malware attack by creating regular backups of valuable information [7]. It is also vital to create user awareness and educate people about the best practices of Mac OS device security [8].

Supervised learning algorithms have also been found to be effective in malicious software detection in the domain of cybersecurity. These algorithms are trained on labeled data to capture distinguishing patterns and features between malware and benign software. Supervised learning models can be trained on such datasets and will be able to generalize on the previous experience and effectively recognize new malware instances [9]. Some of the supervised learning methods such

as Support Vector Machines (SVM), Random Forests (RF), Gradient Boosting (GB), Naive Bayes (NB), and Decision Trees (DT) have been prominently features as among the methods effective in malware classification. Nevertheless, these models need to be constantly updated and re-trained on new malware samples to preserve a high detection rate and keep up with the changing cyber threat landscape. Current research and technology development efforts are making these algorithms increasingly more effective in detecting malware. Supervised algorithms, as the subset of machine learning, expect labeled data to make decisions and predictions, and hence they are suitable when the task is related to classification and regression [10, 11]. They are an important contribution to machine learning in cybersecurity due to their benefits. The effectiveness of the supervised learning models in malware detection is subject to a number of factors such as quality of training data, the number of relevant features to consider, and the regularity of updates to consider new threats. These models should be refined regularly to make them effective in the process of addressing the changing cyber risks [12]. Malicious software is the major security issue that Mac OS X has. Current methods of malware detection on Mac OS X are Host-based Intrusion Detection Systems (HIDS), antivirus, sandboxing, manual forensic investigation, and File Integrity Monitoring (FIM) [13]. Although these methods have some effective points and drawbacks, they are not so capable of detecting the new advanced malware that has never been seen before. An interesting alternative that could be used to detect advanced and new malware threats is supervised Machine Learning (ML) algorithms. These algorithms are used to monitor the behaviors of Mac OS X system to identify malicious behaviors [14]. The ML-based detection requires less time and fewer resources and provides better accuracy than the traditional methods. As Mac OS X has seen a rapid increase in popularity and is now the second most popular operating system after Microsoft Windows attackers have begun to produce malware that specifically attacks the Mac OS X platform.

In this article, we explore the application of supervised learning techniques for detecting and mitigating malware threats in Mac OS X systems. This article seeks to discuss how supervised learning methods can be used to identify and prevent malware risks on Mac OS X environments. The research attempts to improve the reliability, efficiency, and robustness of detecting malware in Mac OS X by examining several classifiers and exploring different feature selection techniques to achieve high detection accuracy. This is also aimed at training an AI-based model that can effectively detect malicious software running on the system. Supervised machine learning algorithms will be implemented to strengthen the current approach to cybersecurity and protect the Mac OS X against new threats. In addition to the technical application, there is a corporate and economical consequence of this research. Enhancing malware detection systems will assist companies and users in reducing the effects of cyber threats and prevent the financial losses linked to security breaches. Furthermore, a more effective detection model would help reduce cyber defense expenditures on enterprises and increase the security of the systems in general. Moreover, the suggested approach will protect vulnerable personal information against unauthorized access, reducing chances of identity theft and financial fraud. This study helps in improving the malware detection thus boosting the security and resilience of the users against dynamic cyber-attacks. The primary contributions of this study are as follows:

- This article proposes to design and develop a supervised learning model for effectively detecting malware in Mac OS X.
- This study aims to determine the optimal supervised model's hyperparameters value to improve Mac OS X's security. A variety of hyperparameters setups are utilized to assess the model's performance.
- We implemented multiple supervised learning techniques and compared them based on their performance metrics. The experiments are conducted using cutting-edge datasets.

This article is organized as follows: Section II discusses related work and comparative analysis. Section III provides a detailed discussion of the research methodology. The results are discussed in Section IV. The conclusion, recommendation, and future efforts are covered in Section V.

II. Related Literature

Malware is any software that is developed to harm files, impair data integrity, steal data, or deny regular system functioning [15]. Because of such threats, it is imperative to note the dangers of such malicious applications and implement proper security policies to prevent the possible infection of Mac OS X systems [16]. On Mac OS X, detection is the initial step toward malware prevention. To achieve this, it is possible to employ the help of special anti-malware programs like Malwarebytes, which is capable of detecting and deleting malware on macOS. Apple has also added a level of security to its users who are not certain whether their system has been infected or not by installing the in-built malware removal tool. This utility is found in applications folder [17]. It is crucial to refrain from deleting any files from the system unless individuals have confirmed that they are associated with the dangerous application. Alternatively, individuals may unintentionally erase crucial system files and thereby inflict further harm upon the system [18]. By adhering to the essential procedures, the individual can safeguard their Mac OS X against infection. Pajouh *et al.* [19] provided a supervised ML approach. This model detects OS X malware using kernel-based SVM and a weight measure based on application library calls. The model was trained and evaluated using 152 malicious and 450 benign instances. The authors used supervised learning techniques and achieved an accuracy rate of 91% and a false positive rate of 3.9%. SMOTE generates more data sets with distinct distributions. This lets us examine how sample sizes affect malware detection accuracy. The authors achieved 96% accuracy and 4% inaccuracy using SMOTE data. Malware categorization uses cross-validation. Research shows that increasing synthetic material sample sizes improves detection accuracy but increases false alarms.

Gharghasheh and Hadayeghparast [20] used multiple ML techniques, including KNN, DT, SVM, and LR, for malware detection Mac OS X. To evaluate the performance and efficiency of their technique, they used ROC Curve. The

author stated library calls enhanced accuracy by 4%, bringing it to 94%. To improve malware detection speed and accuracy, Bensaoud and Kalita [21] propose a multi-task learning framework for the image classification of malware. A DL classifier receives malware-created BMP and PNG files. A new dataset was utilized to evaluate the authors' multi-task learning strategy. The authors collected over 100,000 safe and malicious APK, PE, ELF, and Mach-o files for their study. PReLU averaged 99.87% accuracy in seven tasks and four activation functions. Encryption, Packing, and instruction overlap are all detected by the model. This improves the model's ability to forecast positively and brings it up to the level with the most cutting-edge methods. The ML-based malware detection system MalwD&C allows PE files to be installed safely [22]. The recommended method uses ML classifiers to detect malware in PE files. One public dataset was used to evaluate the MalwD&C approach. In two-class and multi-class classification scenarios, numerous ML classifiers were used. RF outperformed all other classifiers, according to the data. The RF classifier had 99.56% and 97.69% accuracy in two- and multi-class settings. The authors stated that MalwD&C's speed and precision could make it popular in academia and industry. Wardle [23] evaluated Apple's anti-malware procedures, which uncover several flaws before diving into the Mac beginning process. Analyzing malicious OS X apps reveals how code can take advantage of the OS to continue running and surviving reboots. A creative open-source application that lists and shows OS X programs that are persistent and set to launch automatically when the computer is rebooted. With such a solution, users may protect themselves against current and future OS X attacks.

Mercaldo and Santone [24] proposed a method for distinguishing malicious from benign samples in mobile scenarios by determining the malware family to which the sample belongs and the variance within that family. The executable samples are gathered by the authors, who then extract several attributes from the grayscale images. Tested on a dataset of 230 real-world Apple samples and 50,000 real-world Android samples, the approach presented here shows encouraging results, indicating that the method can be applied to produce a wide variety of classifiers. It has been discovered that the proposed approach can distinguish between two variants of a sample, one of which contains an injected payload and the other of which does not. The final values for accuracy and recall in this situation are 0.912 and 0.918, respectively. Table 1 summarizes the current literature.

Table 1. Summary of the existing literature

Ref.	Proposed work	Research Models	Experimental Results
[19]	This work proposed utilizing kernel-based SVM and a novel application library known as the weighting approach to accurately identify OS X malware through supervised ML.	kernel base SVM, SMOTE	The detection accuracy is 91% without SMOTE, with a FAR of 3.9%. However, when SMOTE is used, the detection accuracy increases to 96% with a FAR of less than 4%.
[23]	This study used ML approaches to find malware on Mac OS X and treats library calls as separate features.	SVM, DT, KNN, Ensemble and LR	The Subspace KNN achieved the best accuracy rate of 94.7% percent
[21]	This research provided a new ML method for efficient malware detection.	PreLU	They received a reasonable accuracy rate of 99.87%
[22]	MalwD&C, a ML–based malware detection technique, was created so that users may confidently set up PE files.	RF and MalwD&C Classifier	The RF classifier achieved 99.56 and 97.69 percent accuracy, respectively, in two-class and multi-class scenarios.
[23]	Researchers focused on the starting process of MAC startup in this study.	To survive reboots and remain active, a software code is used	By utilizing the tool, users are assured of their security.
[24]	This study presented a method for differentiating between hazardous and legitimate mobile samples, as well as detecting specific types and variations of malware inside them.	RT, J48, RF, AdaBoost, BN, and DL models	A DNN with ten hidden states achieves an accuracy of 0.918. The supervised ML approaches exhibit varying levels of accuracy, with the RT algorithm obtaining a score of 0.728 and the J48 algorithm obtaining a score of 0.762.

The existing investigation towards the identification of malicious software in Mac OS X is limited to the implementation of purely supervised techniques. While some research has examined the potential of SMOTE techniques to achieve this goal, alternative studies have focused on the implementation of AI (ML) methods and memory forensics. However, scholarly investigations that combine supervised methodologies with an investigation of algorithmic effectiveness for this particular aim are limited. Moreover, the majority of the current research focuses on examining the effectiveness of particular algorithms in identifying malware. This work aims to improve existing cybersecurity mechanisms and protect the system from threats by creating an AI model that can effectively and accurately detect harmful malware in Mac OS X. This will be achieved by utilizing supervised algorithms such as SVM, DT, NB, RF, AB and HGB.

III. METHODOLOGY

Malware detection is commonly perceived as a task of classifying hazardous software. ML techniques are employed to discern between harmful and benign binary data. A DT constructs the strategy by splitting the features using splitting functions. The tree's depth and the splitting criterion (Info gain, Gini index) must be finetuned in the DT method before a classifier can be generated. In order to categorize data, SVM searches for a hyperplane. The hyperplane that divides the random feature space into classes is generated during the training of an SVM classifier using two key parameters: the function of the kernel and the hyperparameters. RF, AB, and GB are all examples of ensemble algorithms that work together to create a more robust classifier. This option is present in some ensemble algorithms and is used to build weak learners using other methods. The histogram-based gradient boosting classifier is used because of its lower

computational cost and improved accuracy when using histogram features. Computing complexity is thus reduced by using the histogram data format. The proposed system architecture can be seen in Fig. 1.

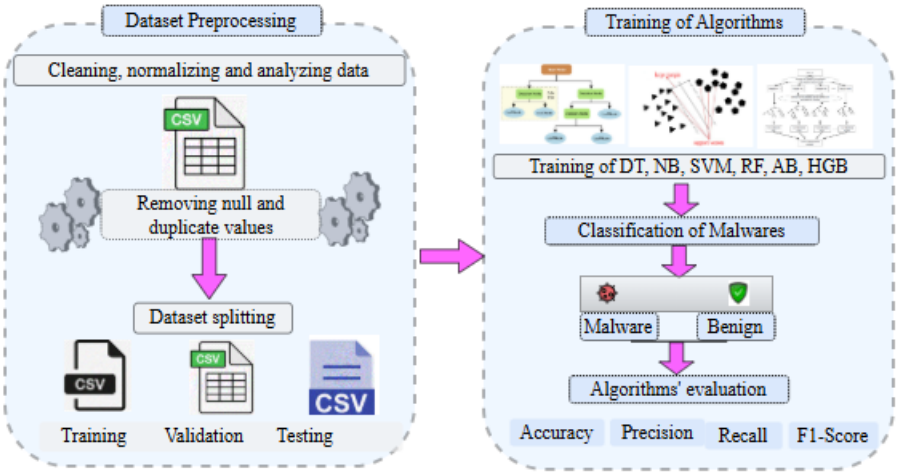

Fig. 1. Proposed system architecture.

A. Implementation Procedure

Table 2 shows the experimental setup for the proposed system.

1. Gather a dataset consisting of documented harmful malware that has been discovered in Mac OS X systems, along with equivalent files that are known to be benign.
2. Prior to further analysis, preprocess the dataset by doing data cleaning, normalization, and data analysis.
3. The training and test sets are created using the selected data set.
4. Train DT, SVC, RF, and NB-supervised algorithms on the training data.
5. Assess the effectiveness of the supervised algorithms on the test set by comparing metrics like precision, accuracy, recall, and F1-Score.
6. Evaluate and understand the findings to ascertain the most efficient algorithm for identifying harmful software in Mac OS X.

Table 2. Experimental setup

Steps	Explanation
Data	Data is collected by the Cyber Science Data Lab. The dataset contains 613 observations and 16 attributes.
Pre-processing	To find and remove the null and duplicate values in the dataset
Data Exploration	Using the Count Plot, Scatter Plot, and Distribution Plot to visualize the input and output attributes
Implementation-Supervised Learning	Supervised Learning Algorithms - DT Classifier, SVM Classifier, RF Classifier, NB Classifier
Algorithms Boosting Algorithms	AdaBoost Classifier, Histogram-based Gradient Boost Classifier

B. Dataset Description and Preprocessing

We have downloaded the dataset from the following link. This dataset will be used in our experiments for all algorithms https://cybersciencelab.com/os-x-malware-dataset/

In 2020, the Cyber Science Lab compiled a collection of harmful and benign files known as the OS X Malware Dataset. Its purpose is to evaluate the efficacy of well-trained algorithms in detecting hazardous malware, specifically on Mac OS X machines. The dataset contains about 6,000 files that can be used on Mac OS X, including both malicious and safe ones. The dataset comprises 152 distinct instances of malicious software. The data was collected from samples obtained during the period from January 2012 to June 2016. viruses, Trojan horses, worms, adware, and spyware are all instances of malicious files. The innocuous data consists of legal programs, system files, and apps. All files have undergone pre-processing and have been categorized as either malicious or benign. Scholars are granted unrestricted access to this dataset to assess the performance of supervised algorithms that were developed specifically for malware detection on Mac OS X. We will, therefore, incorporate this data set into our investigation. This up-to-date and exhaustive compilation of files, encompassing both malicious and benign content, facilitates a more precise evaluation of algorithms. Each file has undergone processing and has been classified as either benign or hazardous. This facilitates the learning process for supervised algorithms.

We perform preprocessing on the data as it is sourced via open-source software. The data we receive contains duplicate entries, null values, and strings that are not useful for data classification. Additionally, there are undesirable columns in the dataset. Preprocessing involves eliminating null and duplicate values. The unique parts are used to figure out if the column is numerical or categorized. We can find the unique element in each column with the nunique() method. The efficiency of the method is tested by dividing the data into groups. The data was split into three separate sets: testing, validation, and training. Fig. 2 shows the count plot for output column “class”.

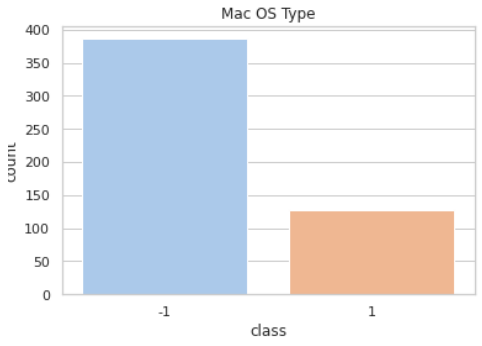

Fig. 2. Count plot for output column “class”.

- **EDA-Mac OS malware data**

Malware for Mac OS is the subject of the data exploration study that follows. This will provide readers with a better visual picture of the data.

The "class" column in the output contains two distinct outputs. The Mac OS type "–1" is considered the majority class due to its widespread usage among users. The minority class is denoted by the Mac OS type "1". Both classes exhibit an imbalance.

The "Segments" column, which is categorical, is shown as a count. According to the data presented in the preceding count plot, segments of type 4 are common in the specified column. Fig. 3 shows the count plot for output column "class".

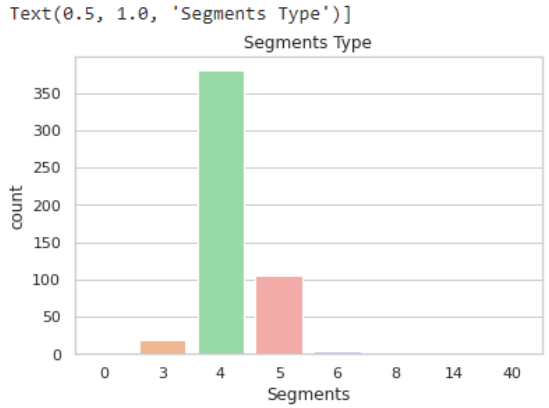

Fig. 3. Count plot for output column "class".

Fig. 4 is a scatter plot depicting the correlation between NCMDS and export_size. The vast majority of NCMDS measurements fall below 100. Most classes are associated with the –1 category, and the file's export_size is much less than 0.2e6.

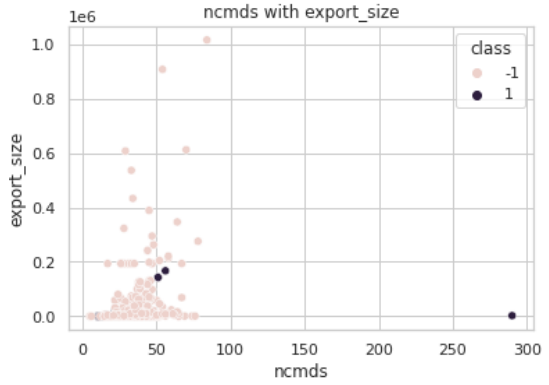

Fig. 4. Scatter plot for "ncmds" with "export_size".

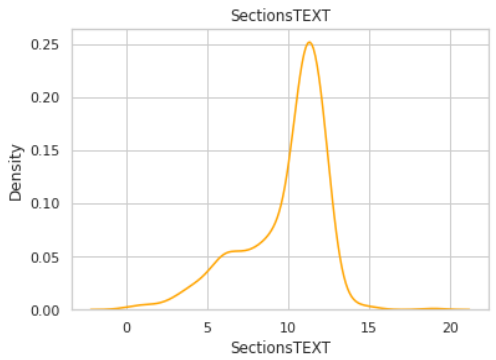

Fig. 5. Displot for "Section TEXT".

A distribution plot for the numerical column "SectionsTEXT" was generated. The numbers in the "SectionsTEXT" column range from 0 to 20. The "SectionsTEXT" variable has a maximum value of 12. Fig. 5 is the Displot for "Section TEXT".

C. Algorithms' Description

The algorithms and the hyperparameters are shown in Table 3.

Table 3. Algorithms and hyperparameters used in this research

Algorithms	Utilized Hyperparameters
SVM	Gamma; Kernel; Tol
DT	Max_features; Max_depth; criterion
RF	Max_features; N_estimators; Max_leaf_nodes
AB	Algorithm; Learning_rate; N_estimators;
NB	Var_smoothing
HGB	Max_bins; Max_iter; Min_simple_leaf

1) Decision tree classifier

A root node, some branches, and some leaf nodes make up DT. At each leaf node, an attribute is checked; at each child node, an attribute result is recorded; and at the last node, the class names are recorded. As its name implies, a tree's root node serves as the origin from which all other nodes branch out. A DT's "nodes" stand in for features (attributes), its "branches" for decisions (rules), and its "leaves" for outcomes (categorical or continuous values) [25, 26]. Fig. 6 is a visual representation of the DT.

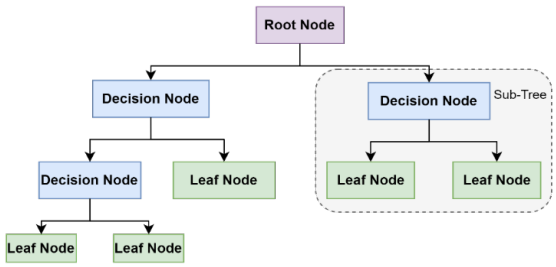

Fig. 6. Decision tree classifier.

DT techniques permit attribute division at any node in order to determine whether dividing is optimal for particular classes. It is imperative that the divisions acquired at each branch remain as free from extraneous influences as feasible. Therefore, it is necessary for the splitting criterion to remain consistent throughout [27]. The training data is divided into multiple smaller groups based on the values of the splitting feature. To ensure that no subsets of instances in any DT do not belong to the same class, the method repeats its recursive steps until this condition is met [28]. The primary advantage of DT is its ability to identify the most biased aspect and its comprehensible nature. Furthermore, it is highly classifiable and comprehensible. Furthermore, it is compatible with both continuous and discontinuous data. When constructing a decision tree, it is enough to evaluate variables and divide the data into different groups based on their characteristics. Non-linear does not adjust any of the DT's parameters based on its performance [29].

- **Hyperparameters**

Max_depth: The utmost vertical distance between the deepest leaf node in the tree and the root node.

Max_features: This parameter determines the maximum number of features to be included when searching for the best

split [30].

Criterion: Refers to a quantitative measure used to assess the feasibility of a branch in a decision tree. Choose the value for Gini Impurity by inputting "gini" or for information gain by inputting "entropy" [31].

2) Support vector machine

A linear predictor is the fundamental component of this system; it is characterized by features that are employed to locate the hyperplane that minimizes the distance between two sets of data. The learning process of SVM involves optimizing a convex quadratic programming problem by expanding the margin. This optimization problem is analogous to minimizing a hinge loss function. Both concerns are synonymous [32]. However, the data usually do not exhibit linear separability. Given these circumstances, it is impossible to find a hyperplane that meets the criterion as it does not exist. The SVM approach suggests selecting a kernel function for addressing non-linear circumstances. Once the calculations in the low-dimensional space are completed, The SVM will transform the input space into a high-dimensional feature space by utilizing the kernel function. Ultimately, the nonlinear data is segregated by constructing an optimal separation hyperplane within a high-dimensional feature space [33, 34]. Fig. 7 shows the Support Vector Machine diagram.

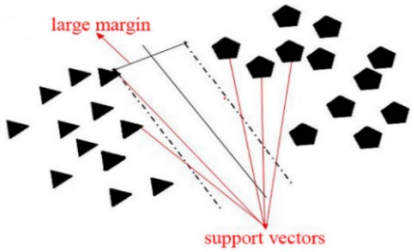

Fig. 7. Support vector machine.

A classification model is proposed, which is based on the concept of hyperplanes to perform linear classification and regression. Within a multidimensional feature space, it defines the boundaries that separate different data instances. The objective of this task is to classify data instances into their corresponding groups [35]. SVM's most valuable characteristics include its versatility as a regression and classification tool, as well as its robustness and precision as a prediction method. The drawbacks of SVM encompass their incapacity to handle multi-class classification, their intricate nature, their inefficiency when dealing with large datasets, and their high memory demands. The SVM classifier is utilized for several purposes, including medical diagnosis and face recognition [36].

- **Hyperparameters**

Kernel: This parameter allows the selection of the kernel type to be used in the algorithm. If no kernel is explicitly defined, the default is assumed to be Radial Basis Function (RBF). Alternatively, a callable function can be provided to compute the kernel matrix directly from the input data [37].

Gamma: The gamma value diminishes as the distance increases. As the gamma value increases, the decision border is determined by examining points that are closer together. Conversely, when the gamma value is lower, the decision boundary is determined by examining points that are further apart.

Tol: This parameter sets the threshold at which the process is considered complete [38].

3) Random forest

RF utilizes a team of DTs in a coordinated manner. DTs are formed through a process of randomly picking pieces from a dataset. The outcomes of decision trees employed in constructing an RF are aggregated by a majority voting mechanism. The advantages of RF include its capacity to handle extensive and intricate information. The method has a strong capability to forecast missing data and ensure accuracy even when a significant amount of data is missing. Additionally, it employs effective strategies to handle mistakes in datasets that have an imbalanced distribution of classes [39]. The crucial aspects of the whole classification are the capacity to accurately predict the characteristics, its versatility for both classification and regression tasks, and the capability to store created forests for future application on different datasets. The RF model has certain drawbacks, such as its relatively higher cost compared to simpler models like DT and its inherent complexity, which poses challenges in interpreting the findings [40, 41]. Fig. 8 shows the process of RF.

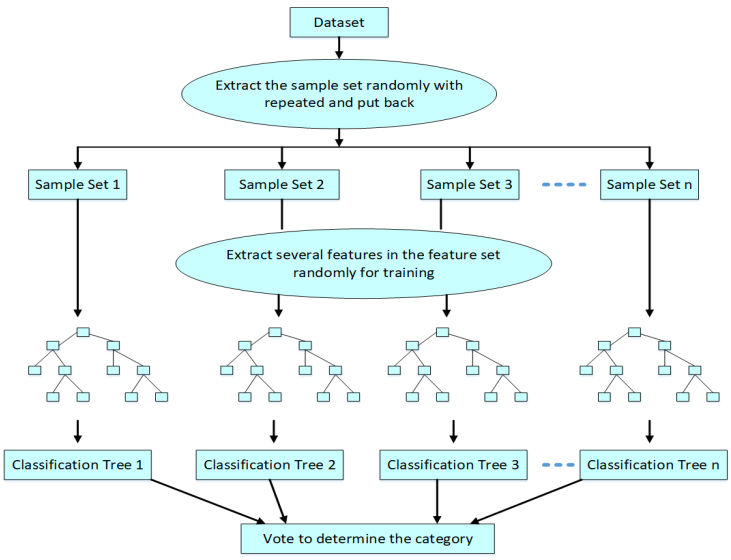

Fig. 8. The process of random forest.

To make a new set of samples using the RF method, "n" random samples are chosen from the first set of samples. After that, these samples are used to make 'n' classification trees. The final choice about categorization is made by taking the average of the results that these trees yield. During tree training, features and data can be chosen at random to see how valuable each one is. As a result of this, the predictor works better after the synthesis process [42].

- **Hyperparameters**

N_estimators: The number of trees in the model's forest can be set by the user with the n_estimators option.

Max_features: It is comparable to the total sum of all possible maximum attributes for each tree in an RF [43].

Max_leaf_nodes: It is a hyperparameter that restricts the growth of the tree by placing a requirement on the division of its nodes [44].

4) Naive bayesian

Naive Bayes can be used for the classification method of assigning labels to issue cases. Training requires less data, which simplifies parameter classification. As linear classifiers, NB classifiers are known for being easy to use and accurate [45]. The benefits of this classifier are its simplicity, speed, effectiveness with both small and large amounts of data, and ease with which correct responses can be obtained. The performance and dependability of the classifier might suffer if the dataset contains a significant number of numerical attributes, which could render the premise of equal relevance and independence invalid [46, 47]. Fig. 9 represents the simple structure of NB.

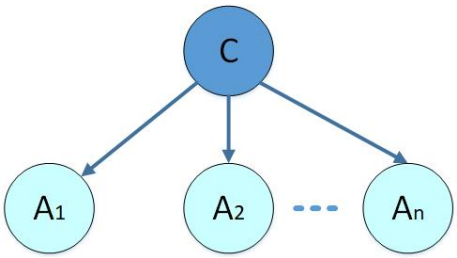

Fig. 9. Structure of naive bayes classifier.

Naive Bayes is often the method of choice when sorting data into categories. The fundamental application of the Bayesian network is the description of a probability distribution using direct acyclic graphs (DACGs). Every "node" in a Bayesian network represents a unique piece of information, and every "arc" between "nodes" represents a specific probability relationship. NB relies on conditional probability in its underlying framework. It builds trees based on how likely each scenario is to materialize. The term "Bayesian Network" is also used to describe these trees [48]. Using this method can result in significant time savings. When there are many different types of predictors, NB can be utilized to find solutions. If the assumption of feature independence holds true, it has the potential to beat competitive models with much less training data [49]. Categorical input variables, as opposed to numeric ones, are ideal for NB. NB makes the assumption that every prediction (or feature) is related, which is obviously not the case. This dramatically complicates the implementation of the strategy. NB classification has found application in a variety of areas, including spam filtering, text classification, and online sentiment analysis [50].

- **Hyperparameters**

Yar_smoothing: This option specifies the weight given to the most significant feature variance in all subsequent calculations [51].

5) Adaptive boosting

This technique employs an iterative process that begins by randomly selecting the training set and, after that, obtaining many subsets. Subsequently, the data is partitioned into subsets to train many foundational classifiers, which are then combined by voting to create a unified and resilient classifier [52]. Conventional predicting systems can provide precise predictions for numerous classes but may overlook certain groups. To address this issue, an AdaBoost algorithm is employed when the distribution of samples is frequently uneven. AdaBoost necessitates the utilization of iterative methods in order to discover a solution. The procedure begins by selecting a random sample as the initial stage. Subsequently, we employ each individual subset of samples to train the foundational classifier. Combining the scores from many classifiers yields a robust final classifier [53]. Fig. 10 depicts the stages of AdaBoost's processing. The DT technique, a multi-node tree system, was used as the base classifier for this investigation. The tree's nodes stand for attribute-based tests, and its branches reflect test outcomes. The final leaf node represents the ultimate outcome, whereas the other nodes each represent a different class. According to Shen *et al.* [54], the sample space is used for both training and evaluating, yielding accurate classification results. Fig. 10 shows the flow of the AdaBoost algorithm.

During weak learning training, AdaBoost initially assigns equal weights to all training data and then adjusts these weights based on the predictions. To reduce bias, the training procedure allocates more weight to the mispredicted samples and less weight to the successfully predicted ones. The weak learners' projected outcomes are linearly combined with weights until the desired number of iterations or anticipated error rate is reached [55].

- **Hyperparameters**

Algorithm: This parameter utilizes two algorithms, namely SAMME and SAMME.R. The efficacy of both criteria is being compared. The SAMME.R algorithm is employed to update the model by considering the probability of estimations. In contrast, SMMEs rely solely on the categorization.

N_estimators: This parameter determines the number of weak learners or simple estimators that are employed in our dataset.

Learning_rate: This parameter is used to decrease the impact of each classifier's contributions. The default value is set to 1 when utilized [56].

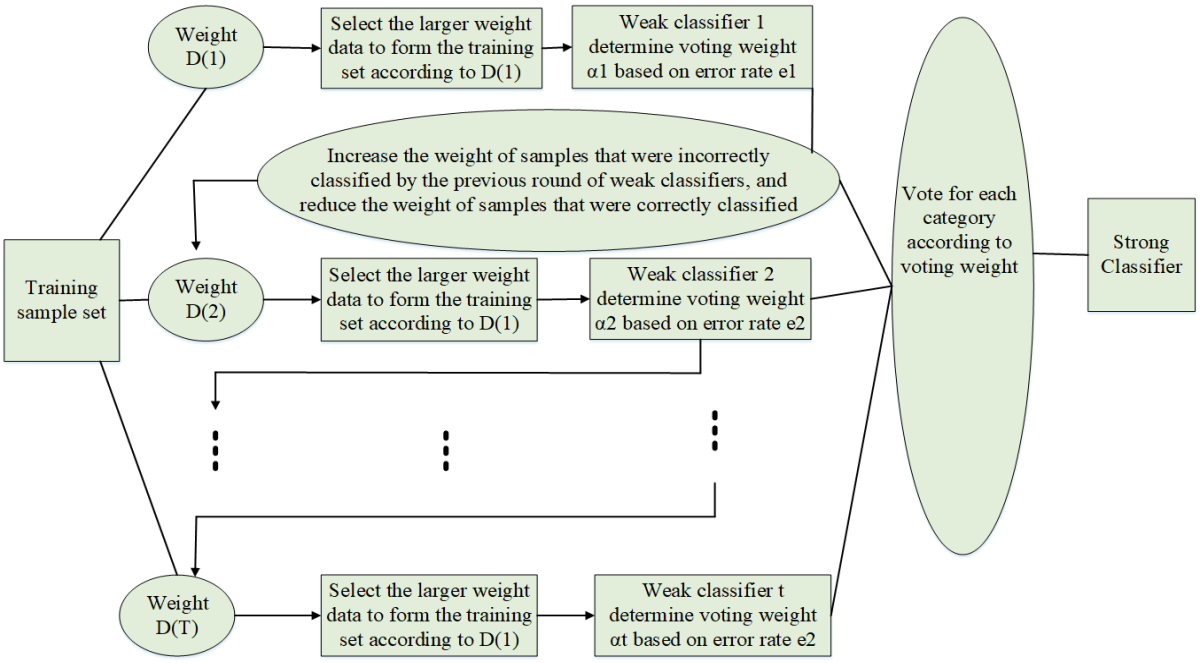

Fig. 10. Flow of the AdaBoost.

6) Histogram Gradient Boosting (HGB)

Recently, this boosting strategy has proven to be highly efficient in improving performance and speed. Boosting strategies amalgamate multiple weak learners into a single entity, resulting in the creation of a resilient and highly efficient strong learner. Initially, the data with assigned weights is utilized to train a model. It is plausible that certain features may frequently appear in the upgraded feature sets of other models [57]. The initial model takes into consideration and enhances the significance of misclassified data samples. The corrected weights are then provided to the second model to refine the problematic predictions. N iterations of this procedure are performed until all n models are operational. Boosting techniques consistently employ a single classifier across all n models, irrespective of whether they employ a DT or another type of classifier [58].

The decision tree is employed in the implementation of the boosting methodology referred to as "gradient boosting." This approach is characterized by its creative nature in addressing classification and regression challenges and has demonstrated promising outcomes across numerous datasets. A significant limitation of gradient boosting is the lengthy duration required for model training [59]. The DT is plagued by an excessive number of features, samples, and inherent flaws. To construct the DT, it is necessary to split or divide all the continuous qualities and their corresponding values. When dealing with several data sets and attributes, this procedure requires a significant amount of time and computational resources. The functionality is degraded as a consequence. The HGB formulates a strategy to address inefficiencies resulting from a voluminous dataset. As implied by its name, this method utilizes a predetermined quantity of histogram bins to partition continuous samples [60]. Fig. 11 shows the HGB converting data into bins.

Employing this method, the space of all possible combinations of feature values is narrowed down to a more manageable subset [58].

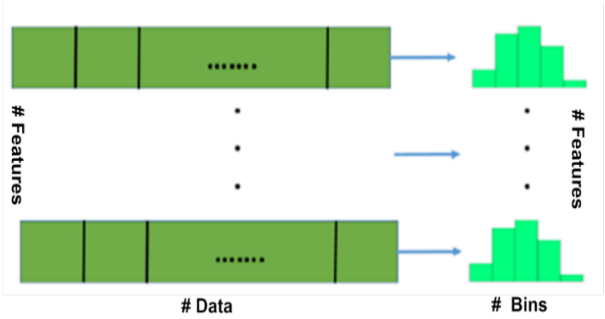

Fig. 11. HGB converting data into bins.

- **Hyperparameters**

Max_iter: The upper limit on the number of iterations for the boosting process, typically referred to as the maximum number of trees utilized for binary classification. During multiclass classification, a total of n_classes trees will be generated with each iteration, as stated by Le *et al.* [61].

Max_bins: The maximum number of containers that can be employed to store non-missing data. Before the training phase begins, each element of the input array X is classified into categories containing whole number values. This facilitates a substantial acceleration in the completion of the training phase. A limited number of distinct values for a feature may necessitate fewer bins than the utmost quantity permitted by max_bins. Aside from the bins designated by max_bins, there is consistently an extra bin allocated for missing data. The maximum allowable value is 255 [12].

Min_samples_leaf: Option determines the minimum number of leaf nodes that must be present at the terminal node.

IV. Metrics Used for Evaluation

Evaluation metrics refer to a systematic approach used to assess and monitor the efficacy of a model. The following metrics are described below.

Precision: It refers to the measure of a model's ability to accurately categorize true positives [62].

Recall: It refers to the ratio of anticipated positive values to the actual positive values in the dataset [62].

F1-Score: It is a metric that takes into account both recall and precision [63].

Accuracy: This metric is utilized to determine the degree to which our model accurately predicts the proper values in its categorization [64].

Speed Efficiency: The precision of a model is assessed by employing separate train and test datasets. The act of partitioning the dataset into two distinct sets is referred to as the train and test methodology [65, 67]. This approach involves utilizing the training set to acquire knowledge about our models, followed by conducting the evaluation on the test set. Upon completion of the testing phase, we can determine the duration required to accomplish the tasks during both the training and testing stages [21].

A. Algorithms' Implementation

In Mac OS X, the existence of hazardous viruses can be detected using two types of algorithms: supervised learning and boosting algorithms [65, 68]. This section reviews the outcomes of the implementations of two classes, each with its own distinct classifier. Fig. 12 shows the algorithms used in Mac OS X.

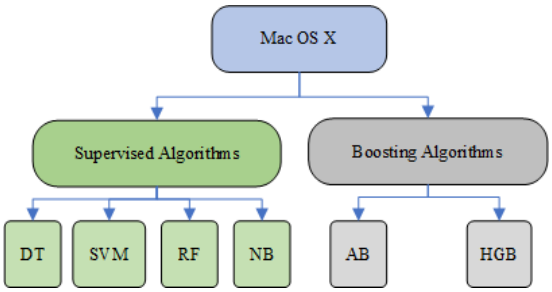

Fig. 12. Algorithms used in Mac OS X.

B. Supervised Algorithms

Supervised algorithms have been categorized into four distinct classes to facilitate the identification of malicious code within Mac OS X. In the following sections, we provide an in-depth evaluation of the four ML algorithms which are discussed in the previous sections, including detailed results and interpretations.

1) Decision tree hyperparameters setup

Table 4 represents the different ML models' hyperparameter setup. Table 4 displays the DT utilized for selecting hyperparameters through a grid search among 45 candidate models, each consisting of 90 fits. The table also includes the corresponding scores, testing time, validation time, and training time for each model. Criteria are set to entropy, Maximum Depth is set to eight, and Maximum Features is sqrt. To detect malware on Mac OS X, the DT model requires 0.187 s for validation, 0.008 s for training, and 0.067 s for testing. The DT achieved a maximum score of 0.80.

2) Support vector machines

Hyperparameters for the SVM classifier are (gamma =s cale, kernel = sigmoid, tol = 0.001), with 40 models and 80 fits displayed in the above table. The final score was 0.75. Detecting hazardous malware on Mac OS X requires 0.056 s for validating, 0.010 s for training, and 0.062 s for testing reasons. The SVM classifier creates a model that can determine whether or not a file is malicious. The system uses a collection of established malware files as a training dataset to acquire knowledge and generate predictions for novel files.

3) Naive bayes

Naive Bayes makes predictions about the other class based on the behavior of the class with the higher probability. The score, validation time, training time, and testing time are all described here, along with an overview of the parameters involved. Table 4 shows the parameters of NB.

In order to detect malicious software on Mac OS X, the hyperparameter var_smoothing was chosen from the NB classifier at a range of 1e−10. The model achieved a score of 0.71 after a validation time of 0.064 s, a testing time of 0.085 s, and a training time of 0.005 s, as shown in the table above. Fig. 13 illustrates the execution time of supervised algorithms.

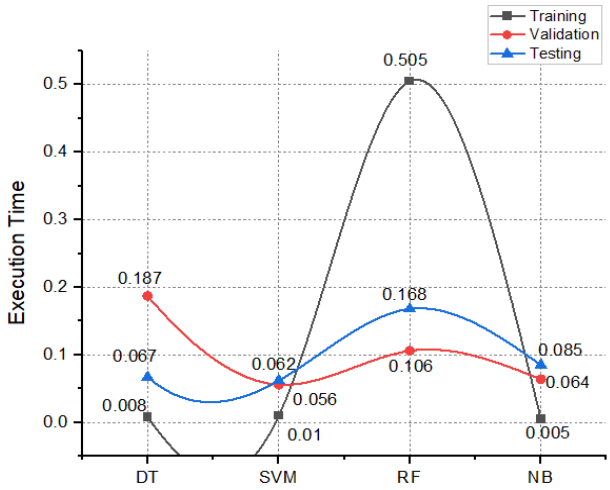

Fig. 13. Execution time of supervised algorithms.

4) Random forest

Table 4 provides an explanation of the hyperparameters utilized in the RF technique, as well as the corresponding score and processing time required for computation. The outcome of the assessment of this algorithm, which involved 90 potential models and 180 iterations, yielded a score of 0.78. When detecting hazardous malware on Mac OS X, the training duration is 0.505 s, the validation duration is 0.106 s, and the testing duration is 0.168 s.

Table 4. Decision tree in Mac OS X

Models	Hyperparameters	Execution Time			
	Utilized Parameters	Score	Testing Time	Training Time	Validation Time
DT	Criterion: entropy, max_features: sqrt max_depth: 8,	0.80	0.067 s	0.008 s	0.187 s
SVM	gamma: scale, tol: 0.001 kernel: sigmoid,	0.75	0.062 s	0.010 s	0.056 s
RF	max_leaf_nodes: 12, max_features: sqrt, n_estimators: 100	0.78	0.168 s	0.106 s	0.505 s
NB	var_smoothing: 1e-10	0.71	0.0850 s	0.064 s	0.005 s

5) Experimental results of the supervised algorithms

The figure below display the assessment metrics for various supervised learning algorithms, DT, SVM, RF, and NB. RF is able to mitigate the issues of variance and overfitting that can arise from using a single DT by combining the predictions of numerous trees. This results in RF achieving superior accuracy, with a testing phase accuracy of 86% and a validation phase accuracy of 94%. Fig. 14 represents the results of supervised algorithms.

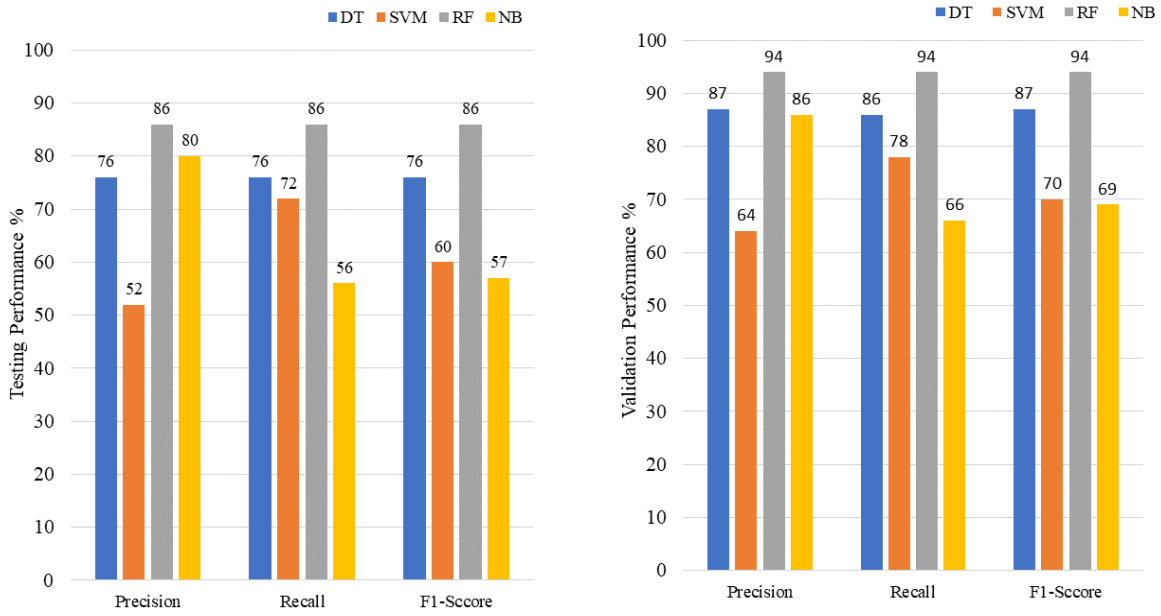

Fig. 14. Results of supervised algorithms.

C. Experimental Result of Boosting Algorithms

Boosting algorithms employ an ensemble approach, combining multiple weak models to create a robust one. Initially, equal weight is assigned to all the examples, and a classifier with low predictive power is developed using the data. Subsequently, the weights of misclassified cases are increased, and the iterative procedure is repeated. There exist two categories of boosting algorithms. This research utilizes Adaboost and histogram-based gradient boost algorithms, which are then contrasted with supervised algorithms.

1) AdaBoost in Mac OS X

The AdaBoost algorithm combines multiple weak classifiers to create a powerful predictor. During each iteration, the algorithm assigns a greater weight to misclassified samples and trains new models using the dataset with adjusted weights. Table 5 shows the AdaBoost hyperparameters.

Table 5. AdaBoost in Mac OS X

Models	Hyperparameters	Execution Time			
	Utilized Parameters	Score	Testing Time	Training Time	Validation Time
AdaBoost	learning_rate: 3.0, algorithm: SAMME, n_estimators: 60	0.75	0.098 s	0.183 s	0.106 s
HBGB	max_iter: 150, max_bins: 250, min_samples_leaf: 20	0.78	1.649 s	0.258 s	0.059 s

Table 5 presents the scores, hyperparameters, and computational time for AdaBoost models. The SAMME method is employed. The learning rate, set at 3, determines the contribution of each weak classifier to the final outcome. Increasing the learning rate enhances the influence of each weak predictor, potentially leading to a more intricate and precise model. The ensemble employs approximately 60 estimators, which are considered to be weak predictors. The score is approximately 0.75. The training process takes 0.183 s, the validation process requires 0.106 s, and the testing process takes 0.098 s.

2) Histogram-based Gradient Boost in OS X

Similar to other boosted tree models, a Histogram-based Gradient Boosting Classifier prioritizes histograms for computational efficiency. Table 5 represents the hyperparameters of the HBGB.

The model parameters utilized are shown in the table above: max_iter = 150, max_bins = 250, and min_samples_leaf = 20. The training time required to detect the malicious program in Mac OS X was 0.258 s, the testing time was 1.649 s, and the validation time was 0.059 s, based on the obtained score of 0.78. Fig. 15 represents the execution time of boosting algorithms.

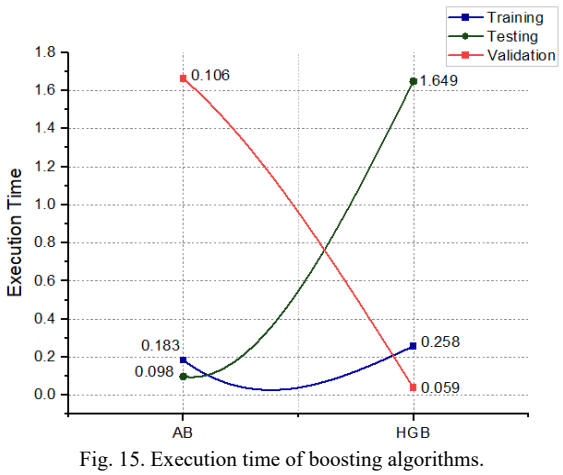

Fig. 15. Execution time of boosting algorithms.

3) Experimental Results of Boosting Algorithms in Mac OS X

Fig. 16 displays the assessment metrics for the boosting methods AdaBoost and histogram-based Gradient Boost. The AdaBoost classifier demonstrates performance improvements in recall, F1-value, and precision: it attains 52%, 52%, and 80% in the testing phase and 61%, 57%, and 85% in the validation phase. Given precision, recall, and F1-Score, the histogram-based Gradient Boost classifier achieves an 89% accuracy in the testing phase and a 91% accuracy in the validation phase. Histogram-based gradient boost is the more effective and scalable classifier for malware detection. Fig. 16 represents the results of boosting algorithms.

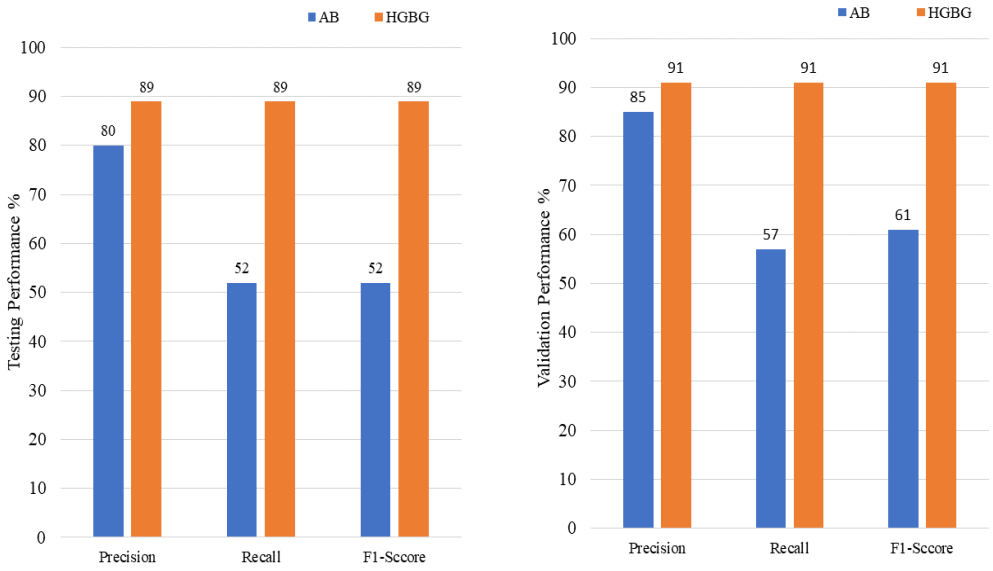

Fig. 16. Results of Boosting Algorithms in Mac OS X.

4) Experimental comparison of boosting supervised algorithms

In this section, we compare various algorithms for detecting malware on Mac OS X to determine which one performs the best. Fig. 17 represents the model's comparison.

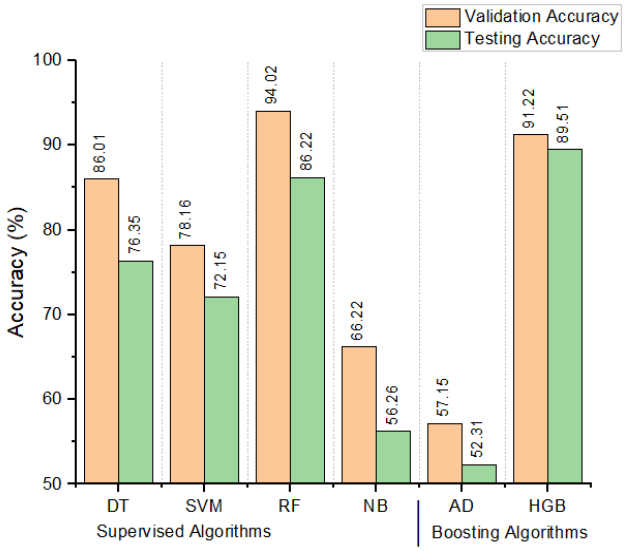

Fig. 17. Algorithm's comparison based on testing and validation accuracy in Mac OS X.

The accuracy of various supervised learning and boosting methods in detecting malicious software on Mac OS X is illustrated in Fig. 17. Every algorithm conducted training using a specific set of data, and subsequently, its correctness was assessed by validating and testing it on distinct data. The percentage values indicate the algorithm's accuracy in correctly categorizing instances of malware. The DT approach achieved obtained 86% accuracy during the validation phase and 76% during the testing phase. Consequently, 86% of the malicious software in the validation data and 76% of the malicious software in the testing data were accurately classified. Similarly, each of the alternative approaches possesses an accuracy score once it completes validation and testing. To provide an example, RF demonstrated a testing accuracy of 86% and a validation accuracy of 94%. On the validation dataset, the NB model attains an accuracy of 66%; on the test dataset, it achieves 56%. It is crucial to bear in mind that accuracy is merely one metric for evaluating the performance of a classifier. Additional metrics, such as the algorithm's recall, precision, and F1-Score, provide valuable insights into the effectiveness of its malware detection capabilities.

V. Conclusion

According to this study, the supervised algorithm's RF classifier and the boosting algorithm's histogram-based boosting are the efficient techniques for detecting malware on Mac OS X. The accuracy of supervised algorithms in validation is 94%, and the boosting algorithm has an accuracy rate of 91%, while in testing, it is 86% and 89%, respectively. Both algorithms demonstrated strong performance, allowing users to choose either one for malware defense. The histogram-based gradient-boosting approach leverages the combination of histogram-based feature extraction and the efficiency of gradient-boosting classifiers. Histogram-based feature extraction structures features into histograms, providing a visual representation of their distribution. This method captures essential characteristics of files, such as byte frequency and the occurrence of specific byte sequences, which helps predict malware behavior. RF has proven to be highly effective in detecting Mac OS X malware with

precision. By aggregating multiple decision trees, RF reduces overfitting and enhances generalization. Additionally, it evaluates feature importance, enabling clear differentiation between malicious and benign software. Handling high-dimensional data is critical for analyzing software files, and RF excels in this aspect. Its ability to process both categorical and numerical data without requiring feature scaling makes it highly adaptable for malware research. Furthermore, RF effectively manages real-world challenges in malware detection, including noise and outliers, making it a robust solution for identifying viruses on Mac OS X.

In conclusion, both the RF classifier and the histogram-based gradient-boosting algorithm have demonstrated acceptable performance in detecting malware on Mac OS X. The achieved accuracy rates were 94% and 91% on validation data, while testing data yielded 86% and 89%, respectively. These results confirm the effectiveness of both algorithms in identifying malicious software with a high degree of confidence. This research holds significant importance for Mac OS X security, as it enhances the protection of users' systems and data against potential cyber threats. Although both models performed well, RF exhibited slightly superior accuracy in both validation and testing phases. The study's findings emphasize the effectiveness of supervised learning and boosting algorithms in malware detection for macOS X. Given its higher accuracy and reliable classification capabilities, RF emerges as the effective supervised model for malware detection, making it a robust choice for strengthening macOS X security. RF attained the maximum accuracy in both validation and testing, 94% and 86%, respectively.

A. Limitations

Due to the use of a single dataset, the conclusions of this research may have been less reliable. Since the research was limited to ML, it is possible that additional algorithms could have produced more substantial outcomes. The impact of continuously updated malicious software was not taken into account in the study. Furthermore, no feature engineering techniques were utilized in this investigation to determine the optimal features for malware detection.

B. Recommendations

The utilization of feature engineering to generate innovative features or sets of features is one of the recommendations for malware analysis. Possible sources for file classification data include file properties, system events, network traffic, and user actions. The file extensions typically associated with malware, the networks malware connects to, and the system events that malware causes can all be used as building blocks for researchers to create features. Ensemble methods are employed to enhance precision by aggregating the predictions of multiple models. Researchers can explore combining the predictions from multiple ML models to enhance overall detection performance. Additionally, DL algorithms offer the potential to further improve accuracy by autonomously learning critical features from raw input data. Convolutional Neural Networks (CNNs) can be employed to extract meaningful features from malware binary code, enabling more precise malware classification and strengthening detection capabilities.

C. Future Work

Improving malware detection on Mac OS X using ML techniques can be further advanced by exploring additional domains. One effective strategy is feature engineering, which involves identifying new features or combinations of existing ones to gain a deeper understanding of malware behavior. This could include analyzing file attributes, system events, network activity, or user behavior to gather relevant data for file classification. Moreover, employing ensemble methods can enhance detection accuracy. Approaches like bagging, stacking, and boosting combine multiple models to achieve better performance in malware detection. Additionally, the integration of advanced AI techniques, such as DL models like or, could significantly improve the system's ability to identify malware. These models are capable of learning critical features directly from raw input data, leading to higher performance and more reliable malware detection.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

IK supervised, reviewed and proofread the article; FW analysed the data, wrote and reviewed the paper; SB conducted the research, and experiments; all authors had approved the final version.

REFERENCES

[1] C. J. Hodson, "Cyber risk management: Prioritize threats, identify vulnerabilities, and apply controls," *Kogan Page Publishers*, 2024.

[2] F. T. Ngo, A. Agarwal *et al.*, "Malicious software threats," in *The Palgrave Handbook of International Cybercrime and Cyberdeviance*, 2020, pp. 793–813.

[3] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Generation Computer Systems*, vol. 97, pp. 887–909, 2019.

[4] A. Shah, B. Ali *et al.*, "Entropy-based grid approach for handling outliers: A case study to environmental monitoring data," *Environmental Science and Pollution Research*, pp. 1–20, 2023.

[5] M. Brundage, S. Avin *et al.*, "The malicious use of artificial intelligence: Forecasting, prevention, and mitigation," arXiv preprint, arXiv:1802.07228, 2018.

[6] I. Kara, "A primary malware analysis method," *Computer Fraud & Security*, vol. 2019, no. 6, pp. 11–19, 2019.

[7] S. H. Han, H. K. Lee *et al.*, "Empirical study on anti-virus architecture for container platforms," *IEEE Access*, vol. 8, pp. 134940–134949, 2020.

[8] I. Nadir and T. Bakhshi, "Contemporary cybercrime: A taxonomy of ransomware threats & mitigation techniques," in *Proc. 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, IEEE, 2018, pp. 1–7.

[9] S. M. Muneer *et al.*, "Cyber Security Event Detection Using Machine Learning Technique," *International Journal of Computational and Innovative Sciences*, vol. 2, no. 2, pp. 42–46, 2023.

[10] X. Gao, C. Hu, C. Shan *et al.*, "Malware classification for the cloud via semi-supervised transfer learning," *Journal of Information Security and Applications*, vol. 55, 102661, 2020.

[11] S. Mahdavifar *et al.*, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of Network and Systems Management*, vol. 30, pp. 1–34, 2022.

[12] S. Kumar, B. Janet, and S. Neelakantan, "Identification of malware families using stacking of textural features and machine learning," *Expert Systems with Applications*, vol. 208, 118073, 2022.

[13] M. Liu, Z. Xue *et al.*, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[14] J. Singh and J. Singh, "Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms," *Information and Software Technology*, vol. 121, 106273, 2020.

[15] A. Case and G. G. Richard III, "Detecting objective-C malware through memory forensics," *Digital Investigation*, vol. 18, pp. S3–S10, 2016.

[16] C. P. Sean, I. Gondal *et al*., "Generative malware outbreak detection," in *Proc. 2019 IEEE International Conference on Industrial Technology (ICIT)*, IEEE, 2019, pp. 1149–1154.

[17] N. Nissim, R. Moskovitch *et al.*, "Novel active learning methods for enhanced PC malware detection in windows OS," *Expert Systems with Applications*, vol. 41, no. 13, pp. 5843–5857, 2014.

[18] C. Xiao and J. Chen, "New OS X ransomware Keranger infected transmission Bittorrent client installer," *Palo Alto Networks Blog*, March 2016.

[19] H. H. Pajouh, A. Dehghantanha *et al.*, "Intelligent OS X malware threat detection with code inspection," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 213–223, 2018.

[20] S. E. Gharghasheh and S. Hadayeghparast, "Mac OS X malware detection with supervised machine learning algorithms," in *Handbook of Big Data Analytics and Forensics*, 2022, pp. 193–208.

[21] A. Bensaoud and J. Kalita, "Deep multi-task learning for malware image classification," *Journal of Information Security and Applications*, vol. 64, 103057, 2022.

[22] B. Khilosiyaa and K. Makadiya, "Malware analysis and detection using memory forensic," *Multidiscip. Int. Res. J. Gujarat Technol. Univ*, vol. 2, no. 2, 106, 2020.

[23] P. Wardle, "Methods of malware persistence on Mac OS X," in *Proc. the Virus Bulletin Conference*, 2014.

[24] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 157–171, 2020.

[25] H. H. Patel and P. Prajapati, "Study and analysis of decision tree-based classification algorithms," *International Journal of Computer Sciences and Engineering*, vol. 6, no. 10, pp. 74–78, 2018.

[26] B. Charbuty and A. Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, no. 01, pp. 20–28, 2021.

[27] Priyanka and D. Kumar, "Decision tree classifier: A detailed survey," *International Journal of Information and Decision Sciences*, vol. 12, no. 3, pp. 246–269, 2020.

[28] K. Yadav and R. Thareja, "Comparing the performance of naive bayes and decision tree classification using R," *International Journal of Intelligent Systems and Applications*, vol. 11, no. 12, 11, 2019.

[29] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends, and challenges," *Journal of Network and Computer Applications*, vol. 153, 102526, 2020.

[30] E. Spiliotis, "Decision trees for time-series forecasting," *Foresight*, vol. 1, pp. 30-44, 2022.

[31] R. G. Mantovani, T. Horváth *et al.*, "An empirical study on hyperparameter tuning of decision trees," arXiv preprint, arXiv:1812.02207, 2018.

[32] Y. Zhou, P. Ram *et al.*, "Flora: Single-shot hyper-parameter optimization for federated learning," arXiv preprint, arXiv:2112.08524, 2021.

[33] J. Cervantes, F. Garcia-Lamont *et al.*, "A comprehensive survey on support vector machine classification: Applications, challenges, and trends," *Neurocomputing*, vol. 408, pp. 189–215, 2020.

[34] F. Wahab, A. Shah, I. Khan, B. Ali, and M. Adnan, "An SDN-based Hybrid-DL-driven cognitive intrusion detection system for IoT ecosystem," *Computers and Electrical Engineering*, vol. 119,109545 2024.

[35] H. HaddadPajouh, A. Dehghantanha, *et al.*, "Intelligent OS X malware threat detection with code inspection," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 213–223, 2017.

[36] L. Gigović, H. R. Pourghasemi, S. Drobnjak, and S. Bai, "Testing a new ensemble model based on SVM and random forest in forest fire susceptibility assessment and its mapping in Serbia's Tara National Park," *Forests*, vol. 10, no. 5, 408, 2019.

[37] C. R. Vishwanatha, V. Asha *et al.*, "Support Vector Machine (SVM) and Artificial Neural Networks (ANN) based Chronic Kidney Disease Prediction," in *Proc. 2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, 2023, pp. 469–474.

[38] I. Roman, R. Santana *et al.*, "In-depth analysis of SVM kernel learning and its components," *Neural Computing and Applications*, vol. 33, no. 12, pp. 6575–6594, 2021.

[39] N. S. Bajaj, A. D. Patange *et al.*, "Application of metaheuristic optimization-based support vector machine for milling cutter health monitoring," *Intelligent Systems with Applications*, vol. 18, 200196, 2023.

[40] Z. Khanam, B. N. Alwasel *et al.*, "Fake news detection using machine learning approaches," *IOP Conference Series: Materials Science and Engineering*, vol. 1099, no. 1, 012040, IOP Publishing, 2021, March.

[41] E. K. Sahin, "Assessing the predictive capability of ensemble tree methods for landslide susceptibility mapping using XGBoost, gradient boosting machine, and random forest," *SN Applied Sciences*, vol. 2, no. 7, 1308, 2020.

[42] S. Wei, L. Zhu *et al.*, "An Adaboost-based intelligent driving algorithm for heavy-haul trains," *Actuators*, vol. 10, no. 8, 188, 2021, August.

[43] X. Tan, S. Su *et al.*, "Wireless sensor networks intrusion detection based on SMOTE and the random forest algorithm," Sensors, vol. 19, no. 1, 203, 2019.

[44] J. Korstanje, "The random forest," in *Advanced Forecasting with Python: With State-of-the-Art-Models Including LSTMs*, Facebook's and Amazon's DeepAR, Berkeley, CA: Apress, 2021, pp. 179–191.

[45] A. Yokoyama and N. Yamaguchi, "Optimal hyperparameters for random forest to predict leakage current alarm on premises," in *E3S Web of Conferences*, vol. 152, 03003, EDP Sciences, 2020.

[46] I. Wickramasinghe and H. Kalutarage, "Naive Bayes: Applications, variations, and vulnerabilities: A review of literature with code snippets for implementation," *Soft Computing*, vol. 25, no. 3, pp. 2277–2293, 2021.

[47] P. Agrawal and B. Trivedi, "Machine learning classifiers for Android malware detection," in*: Proc. ICDMAI, Data Management, Analytics, and Innovation*, Springer Singapore; 2020, pp. 311–322.

[48] V. Kadam, S. Kumar *et al.*, "Enhancing surface fault detection using machine learning for 3D printed products," *Applied System Innovation*, vol. 4, no. 2, 34, 2021.

[49] R. P. Kannan, "Prediction of consumer review analysis using Naive Bayes and Bayes Net algorithms," *Turkish Journal of Computer and Mathematics Education (TURCOMAT),* vol. 12, no. 7, pp. 1865–1874, 2021.

[50] S. Manimurugan, "IoT-Fog-Cloud model for anomaly detection using improved Naïve Bayes and principal component analysis," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2021.

[51] S. Ray, "A quick review of machine learning algorithms," in *Proc. 2019 International Conference on Machine Learning, Big Data, Cloud, and Parallel Computing (COMITCon)*, IEEE, 2019, pp. 35–39.

[52] E. Elgeldawi, A. Saye *et al.*, "Hyperparameter tuning for machine learning algorithms used for Arabic sentiment analysis," *Informatics*, vol. 8, no. 4, 79, 2021, November.

[53] G. Algan and I. Ulusoy, "Image classification with deep learning in the presence of noisy labels: A survey," *Knowledge-Based Systems*, vol. 215, 106771, 2021.

[54] F. Shen, X. Zhao *et al.*, "A novel ensemble classification model based on neural networks and a classifier optimisation technique for imbalanced credit risk evaluation," *Physica A: Statistical Mechanics and its Applications*, vol. 526, 121073, 2019.

[55] I. Ahmad, "An efficient network intrusion detection and classification system," *Mathematics*, vol. 10, no. 3, 530, 2022.

[56] S. M. Basha, D. S. Rajput, and V. Vandhan, "Impact of Gradient Ascent and Boosting Algorithm in Classification," *International Journal of Intelligent Engineering & Systems*, vol. 11, no. 1, 2018.

[57] R. Krithiga and E. Ilavarasan, "Hyperparameter tuning of AdaBoost algorithm for social spammer identification," *International Journal of Pervasive Computing and Communications*, vol. 17, no. 5, pp. 462–482, 2021.

[58] M. Ibrahim, H. AbdelRaouf *et al.*, "Keystroke dynamics-based user authentication using Histogram Gradient Boosting," *International Journal of Computers and Information*, vol. 10, no. 1, pp. 36–53, 2023.

[59] S. M. M. Hossain and K. Deb, "Plant leaf disease recognition using histogram-based gradient boosting classifier," in *Proc. the 3rd International Conference on Intelligent Computing and Optimization 2020 (ICO 2020)*, Springer International Publishing, 2021, pp. 530–545, January.

[60] A. Anghel, N. Papandreou, T. Parnell *et al.*, "Benchmarking and optimization of gradient-boosting decision tree algorithms," arXiv preprint, arXiv:1809.04559, 2018.

[61] T. Le, B. Vo, H. Fujita *et al.*, "A fast and accurate approach for bankruptcy forecasting using squared logistics loss with GPU-based extreme gradient boosting," *Information Sciences*, vol. 494, pp. 294–310, 2019.

[62] L. Zhu and P. Spachos, "Support vector machine and YOLO for a mobile food grading system," *Internet of Things*, vol. 13, 100359, 2021.

[63] F. Wahab, I. Khan, T. Hussain, and A. Amir, "An investigation of cyber-attack impact on consumers' intention to purchase online," *Decision Analytics Journal*, 100297, 2023.

[64] Y. M. Chen, C. H. Yang, and G. C. Chen, "Using generative adversarial networks for data augmentation in android malware detection," in *Proc.*

2021 IEEE Conference on Dependable and Secure Computing (DSC), IEEE, 2021, pp. 1–8.

[65] R. Bapat, A. Mandya, X. Liu *et al.*, "Identifying malicious botnet traffic using logistic regression," in *Proc. 2018 systems and Information Engineering Design Symposium (SIEDS),* IEEE, 2018, pp. 266–271.

[66] F. Wahab, S. Ma, X. Liu, Y. Zhao, A. Shah, and B. Ali, "A ranked filter-based three-way clustering strategy for intrusion detection in highly secure IoT networks," *Computers and Electrical Engineering*, vol. 127, 110514, 2025.

[67] F. Wahab, I. Khan, and K. Si, "Investigating offline password attacks: A comprehensive review of rainbow table techniques and countermeasure limitations," *Romanian Journal of Information Technology and Automatic Control*, vol. 34, no. 1, pp. 81–96, 2024. https://doi.org/10.33436/v34i1y202408

[68] W. Fazal, S. Ma, Y. Zhao, and A. Shah, "An explainable three-way neural network approach for intrusion detection in IoT ecosystem," *Internet of Things*, vol 33, 101722, 2025