RESEARCH ARTICLE

INTERNATIONAL RESEARCH JOURNAL OF MULTIDISCIPLINARY TECHNOVATION

Asian Research Association

DOI: 10.54392/irjmt25616

Comparative Evaluation of Machine Learning Algorithms for Efficient Malware Detection

K. Vijayalakshmi [a,*] Eshaa Balamurugan Subhashini [a], Safiya Al Sabahi [a], Rahmah Al Shanawi [a], Tahani Al Harthy [a]

[a] College of Engineering, National University of Science and Technology, Muscat, Sultanate of Oman.

* Corresponding Author Email: vijayalakshmi@nu.edu.om

DOI: https://doi.org/10.54392/irjmt25616

Received: 26-03-2025; Revised: 08-11-2025; Accepted: 17-11-2025; Published: 30-11-2025

Abstract: Malware is becoming a severe threat globally, affecting many industries. With the growing number of new malware variants every day, the traditional signature-based methods cannot cope and fail to detect them. Hence, there is a serious need for more effective systems to detect malware and protect people from facing losses due to malware attacks. This paper aims to detect malware using machine learning algorithms and compare the results. The appropriate dataset was collected and cleaned by encoding non-numeric data. The correlation-based feature selection method was applied to choose the essential features and drop the independent features that were highly correlated with each other. The dataset was then divided into training and testing sets. Different algorithms like Random Forest, Naïve Bayes, Logistic Regression, and Support Vector Machine were trained using the training set to build the machine learning model. The models were then tested using the testing set. Different parameters like accuracy, true positive rate, false positive rate, F1-score, precision, ROC curve, and confusion matrix were used to analyze and compare the performance of each algorithm. Random Forest scored the highest accuracy of 99%, followed by Support Vector Machine with an accuracy of 94%. In contrast, Naïve Bayes and Logistic Regression performed poorly, achieving only 63% and 52% accuracy, respectively. The results of this research will enhance the strength of cybersecurity systems through advanced malware detection. It will help protect businesses and individuals from the risk of exposing sensitive information, such as identity documents, financial statements, and confidential proprietary information. In addition, it may avoid operational interruptions, harm to reputation, loss of data, and massive financial damage that may result from ransomware, corporate spying, and intercontinental service interruptions.

Keywords: Malware detection, Correlation-based feature selection, Logistic Regression, Naïve Bayes, Random Forest, Support Vector Machine

1. Introduction

Malware, or malicious software, is a type of software that is maliciously developed by cybercriminals to damage, harm, or misuse services, programmable devices, or networks. It is a persistent threat in the contemporary digital landscape as it has the potential to encrypt, steal, or delete information, disable critical computer functions, and monitor users without consent [1]. Ransomware, worms, viruses, and trojans are some of the commonly known cyber malware, and they exhibit unique behaviors and attack patterns [2].

Malware cyber-attacks have historically brought some dire consequences including the breaching of sensitive information, incurring financial losses, eroding customer trust, business shutdowns, and a negative impact on the larger economy [3]. Recent malware variants are reported to evolve quite rapidly, employing sophisticated and stealthy techniques that defy traditional security models [4].

Reliable software without failure is mandatory for all software solutions, including real-time systems [5]. Software failure or damage may cause potential loss to the user and create a hazardous environment for the system and humans in real-time software-based hyperphysical systems [6]. So, Effective detection systems are critical, as failure to do so consistently risks the global cost of cybercrime exceeding $12.2 Trillion by the year 2031 [7]. Earlier studies already highlighted that the economic cost of cyberattacks is expected to grow into the multi-trillion-dollar range by 2025 [8]. Older, signature-based systems are struggling to detect ever evolving mutated malware variants and zero-day

threats. There is a dire need to shift to more flexible, intelligent systems that address this gap.

In this way, the use of Artificial Intelligence (AI) and Machine Learning (ML) technologies has become increasingly important in the field of cyber security. Compared to the traditional static, rule-based approaches, they can identify anomalies and patterns, as well as new and novel forms of malware, with far greater precision and adaptability. It has been demonstrated that ML approaches such as the Random Forest, SVM, and even neural networks have positively impacted the frameworks for detection of threats and enabled proactive defense mechanisms [9].

Many recent studies on Malware highlight the research gap in the proposed field. The existing systems are less robust and need a solution to give more confidence in the system. Another challenge is adversarial evasion. Small perturbations to binaries or behavioral traces can bypass many deep learning detectors. Some literature supports CNN, RNN, and Transformers, which will give superior performance, but creates a black box problem, which reduces trust for cybersecurity. Also, real-time deployment is limited and creates a high computational cost. Bensaoud et al did a literature review on deep learning approaches and emphasized the need for adversarial robust static feature analysis and dynamic behaviors [10]. Pinhero et al proposed visualization-based malware detection where binaries were converted into images for a CNN. The authors claimed this methodology has improved detection accuracy on polymorphic malware [11]. Hussain et al analysed ransomware defense using hybrid deep learning, highlighting that real-time applicability includes scalability issues [12].

The specific objectives of the paper are as follows.

1. To develop an ML model using Python programming to detect diverse malware families
2. To identify the robustness of the ML models in detecting malware families against adversarial and obfuscation techniques
3. To design and implement an interface that visualizes enhanced detection results, interpretability, and decision support
4. To compare the proposed methodology with other similar methodologies proposed by previous researchers.

1.1 Novelty of the proposed work

In addition to the work that has been accomplished regarding the use of Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes (NB), and Logistic Regression (LR) classifiers for malware detection, the current work brings new contributions through the incorporation of correlation feature selection on the system-behavior attributes. Features indicative of malware processes include static_prio, map_count, end_data, and free_area_cache, which reflect process memory allocation, process prioritization, and execution pattern. Relevant feature selection and pruning of redundant features accomplish the proposed goal of maintaining high predictive accuracy while also simplifying the model, which expunges unnecessary computational overhead. It also ensures the model's flexibility for close to real-time detection, as fewer features will accelerate the processing of incoming files. System behavior features also improve the resistance against adversarial evasion whereby malware alters its behavior to evade detection. Because malware processes are typically cumbersome, it becomes inevitable to control low-level system behavior attributes without revealing its presence. These improvements, as a whole, provide high practical value given modern needs for effective malware detection while moving beyond simple comparisons of classifier performance.

2. Literature Review

2.1 Algorithm Types

A method was proposed for classifying malware using five different machine learning classifiers, and an appropriate model was found for categorising encrypted communications. Grid search is used to fine-tune the hyperparameters, significantly enhancing their models' performance. The MTA-KDD'19 dataset created by Ivan Letteri *et al.* is used as the dataset. With an accuracy of 99.27%, the Random Forest classifier performed better than all other models. Strong results were also produced with the Support Vector Machine classifier, with performance metrics above 0.9. However, on the other hand, the random forest can be too slow and inefficient for real-time forecasts when there are many trees [13].

Akhtar & Feng used various classifiers like K-Nearest Neighbors (KNN), Convolutional Neural Networks (CNN), Naïve Bayes, Random Forest, Support Vector Machine (SVM) and Decision Tree (DT). KNN, CNN, Naïve Bayes, Random Forest, SVM, and DT to improve computer network security by detecting harmful traffic and achieved accuracy results of 95.02%, 98.76%, 89.71%, 92.01%, 96.01%, and 99% respectively. The data used was provided by the Canadian Institute for Cybersecurity, which contained log data for various types of malware. 51 distinct malware families are found and tackled the problem of large datasets by building a smaller set of features from a more extensive set. However, the DT algorithm has a scope of overfitting [14].

A study was made on how machine learning can identify malware using four different algorithms: decision tree, Gradient Boosting, Random Forest, and SVM. A flexible framework is presented with more than 130000

datasets, and many features are extracted from binary files to train and test the algorithms. The algorithms are evaluated based on accuracy, execution time, false positives and negatives rate, and area under the Receiver Operating Characteristic curve. Random Forest performed the best with an accuracy rate of more than 99.64% and a 0.9933 area under the ROC curve [15].

A classification model was created to separate the files into malicious and non-malicious ones. Open data sources were used to gather the dataset for this investigation. They used five algorithms: Naive Bayes, Random Forest, Gradient Boost, XGBoost, and Decision Tree. The result was achieved successfully and works well as a screening tool, and the system can be enhanced by adding methods like signature and anomaly-based detection. This research identified the best methods for classifying the individuals using the dataset, such as Random Forest and XG Boost, which have around 99.95% accuracy. The drawback of the research is that the usage of fewer datasets works well as a screening tool, and the system can be enhanced by adding methods like signature and anomaly-based detection. The best methods for classifying the individuals using the dataset are Random Forest and XG Boost, which have around 99.95% accuracy. The drawback of the research is the use of fewer datasets.

A system was developed using machine learning classifiers such as K-Nearest Neighbor, Support Vector Machine, Decision Tree, Neural Networks, Random Forest, and Naive Bayes to detect and identify malware through dynamic and static analytics. Cuckoo Sandbox is used for malware behaviour extraction and tests the classifiers with processed datasets. The Random Forest algorithm achieved the highest accuracy of predictions for API call sequences and PE imports, reaching 98.89% with the dynamic dataset and 98.17% with the static dataset [16].

Zhao, Chen, and Wu [17] created a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network-based hybrid deep learning model for malware detection by using opcodes sequenced from executable files. For spatial information, the CNN processes sections of opcode data and the temporal information is extracted by LSTM. The hybrid CNN–LSTM model outperformed standalone CNN or LSTM models, both in accuracy and F1-scores, which validates the usefulness of the spatial and temporal feature learning techniques for malware analysis.

2.2. Feature Selection and Optimization

The importance of feature selection lies in improving the model's performance through the reduction of dimensions and eliminating unnecessary features. This also helps in improving the generalization of the model. The critical feature selection is more important for performance optimization of software-based systems [18]. In the feature selection, it is required to analyse the features of each software component to have better reliability, better performance and failure free [19]. Akhtar and Feng [14] tackled the problem of high-dimensional data by forming a subset of features from a larger set. This method helped in faster training, decreased noise, and maintained classifying accuracy. This approach was useful particularly for the Random Forest and CNN models.

In a separate study, Masum *et al.* [15] automated feature selection for ransomware detection using the variance threshold and variance inflation factor (VIF). The variance threshold method helps remove features which have the same value across samples because such features lack the ability to predict outcomes. In the meantime, the VIF was used to find and remove features that have multicollinearity, a situation in which independent variables are highly correlated among themselves. This is critical because multicollinearity creates a biased estimation of coefficients which is an obstacle in determining how each feature individually affects the model.

Their findings proved that model accuracy and interpretability were enhanced using those techniques, especially for ensemble classifiers like Random Forest, which tend to be overloaded by irrelevant features. Additionally, the techniques improved the training time and tailored the models for real time malware detection.

Our motivation stems from these findings, which inspired us to apply a correlation-based filter technique that works by finding and eliminating features that are too strongly associated with each other, thus introducing the potential risk of overfitting or model instability. Even though our methods are not precisely the same, they work with the primary aim of increasing the robustness and interpretability of the models by reducing the dimensionality of the feature space.

2.3. Specialized Techniques and Deep Learning Approaches

A study was conducted to improve computer system security and malware detection. Hardware Performance Counters (HPC) methods with different algorithms, like Neural Networks, are used. The best results are achieved with the Convolutional Neural Network (CNN) algorithm with 99.69% and 98.22% accuracy for training and testing, respectively. Full Order Radial Basis Function (RBF) algorithm performed marginally better than CNN and found that branch instructions, instructions, branch misses, and cache references are the most frequently used HPC characteristics. J48, JRip, MLP, and OneR are the most used algorithms, while CNN, BayesNet, MLP, and J48 achieve the best average accuracy outcomes [16].

A deep ensemble technique was developed that integrates CNN with LSTM, yielding more than 98%

accuracy on recent malware detection tasks [17]. In parallel, an analysis of the performance of transformer models for dynamic malware classification demonstrated increased efficacy against changing threat landscapes [20]. Additionally, a study used federated learning for privacy-preserving malware detection across decentralized networks. Such works build upon this research and underscore the possibilities of using machine learning to strengthen cybersecurity [21].

New work has been done to improve the detection of malware. Shokouhinejad discussed graph learning and explainability and how they add value to detection systems [22]. Gond and Mohapatra analyzed the problem of concept drift using a deep learning model with genetic algorithms [23]. An investigation was made on hybrid deep learning techniques for real-time malware detection [24]. In embedded systems, a hybrid model was proposed based on CNNs and transformers for efficient malware detection [25].

2.4. Ransomware-Specific Detection

A study on ransomware detection methods from 2019 to 2021 found that dynamic analysis has the highest detection accuracy rate. Three aspects related to ransomware studies are identified: Red Flags Reflecting the Occurrence of Ransomware Attack, Ransomware Attack Model, and Ransomware Attack Behavior. The study aims to provide a user guide to encourage researchers to use available techniques to detect ransomware attacks. However, static analysis has limited scope due to its limited accuracy and high false alarms [15].

2.5. Domain-Specific Dataset Applications

A research presented on PDF Malware Detection based on Decision Trees with Evasive-PDFMal2022 dataset, which comprises 10,025 records of benign and malicious PDF files with 37 static features. The AdaBoost algorithm is employed to build a decision tree model with various hyperparameter options. The results showed a classification accuracy of 98.7% [26].

The studies analyzed so far seem to agree on one point, machine learning and deep learning algorithms like Random Forest, CNN, SVM, and even hybrid models are capable of performing with high accuracy in malware detection across multiple datasets and scenarios. Accuracy was further augmented by feature selection techniques as well as dynamic and static analysis, and the application of sandbox environments. Regardless, most of these works focus on the empirical accuracy of models and overlook the measures of scalability, adaptability to concept drift, or convenient real-time deployment. Some methods overemphasise results derived from narrow or domain-specific datasets, which expands the scope of the limited generalizability of the results. Also, the explainability of model decisions, which is paramount for the cybersecurity end-users, is inadequately answered. To address the above-mentioned gaps, this work attempts to design a highly accurate yet efficient explainable and adaptive malware classification framework that adjusts to changes in malware behaviors.

3. Methodology and Implementation

Figure 1 shows the block diagram of the methodology of the work carried out. Initially, the appropriate datasets are collected, containing malware and clean files. One of the most crucial steps is to clean the datasets because regardless of how good an algorithm works; it is only possible to obtain good results from good data. Then, feature selection techniques are applied to choose the essential features. The dataset is then divided into training and testing sets. The training set is trained with the different machine learning models, and the testing set is tested using the built models to detect the malware and clean files. All machine learning models, including SVM, were trained and tested using the same training and testing dataset split to ensure a fair and consistent comparison of performance metrics.

Hyperparameter Tuning: As part of the classification improvement processes, hyperparameter tuning was performed on SVMs and Random Forest classifiers. In SVM, a kernel of type linear and rbf was searched for along with the regularization parameter C using grid search. The highest accuracy for SVM was obtained when the RBF kernel was used and C was set to 1.0.

For the Random Forest model, parameters such as the number of trees N and the maximum depth of each tree M were tuned. The best results were achieved with N set to 100 and M set to None. Parameter tuning was done with cross-validation in Scikit-learn using GridSearchCV, which enhanced the model's selection.

This statement further expands as follows: Hyperparameter optimization was carried out on Random Forest (RF) and Support Vector Machine (SVM) for classification performance and reproducibility. For Random Forest, the number of estimators parameter was set between 50 and 200, the max depth was set between 10 and None, and the minimum samples leaf was set between 1 and 5. A 5-fold cross-validation using GridSearchCV of scikit-learn was employed, resulting in maxdepth of None and number of estimators equaling 100. For SVM, the types of kernels (linear and rbf), C parameter for regularization (between 0.1 and 10), and gamma (scale, auto) were adjusted using 5-fold GridSearchCV which resulted with the best settings as the RBF kernel, C equalling 1.0, and gamma equalling 'scale'. All experiments were conducted on an Intel Core i7-10750H CPU with 16 GB RAM for reproducibility and for clearly documenting the experiments in the proposed studies.

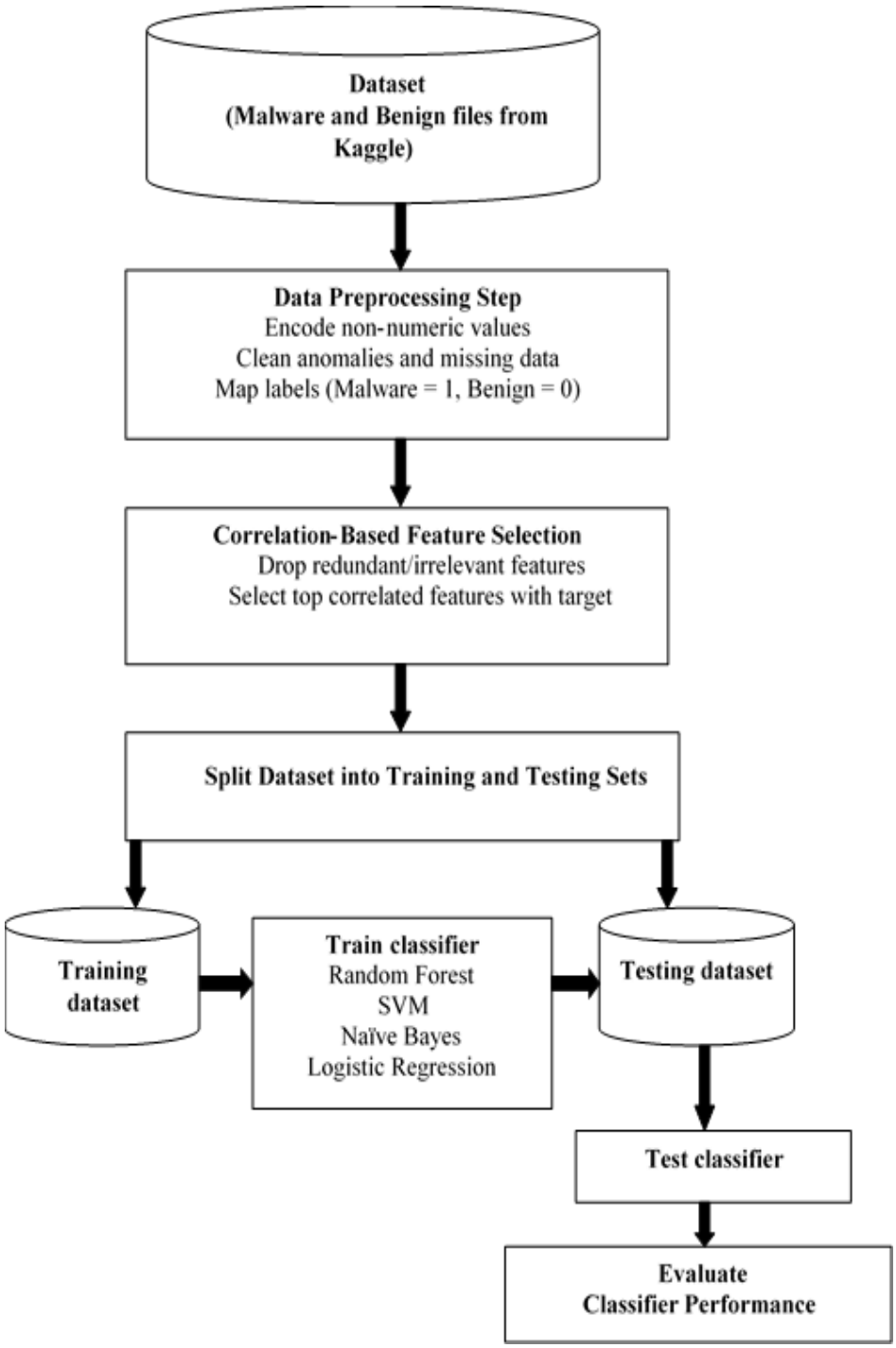

Figure 1. Block Diagram of Malware Detection Using Machine Learning Algorithms

3.1 Dataset

The dataset was collected from the Kaggle website. The dataset used has been created using the network traffic of a Unix/Linux-based virtual machine solely for detecting malware and classification purposes. It contains 50,000 malware files and 50,000 clean files with 35 features [27]. Table 1 describes the features present in the dataset. Figure 2 shows the distribution of the classes in the dataset in the form of a bar graph. Figure 3 shows the different features and their values present in the dataset.

3.2 Cleaning Dataset

The dataset is then cleaned by encoding non-numeric data. The numeric values 0 and 1 are mapped to the classification column's 'benign' and 'malware' values based on the mapping dictionary. Most machine learning algorithms work best with numerical data, so converting data to this format is common. Additionally,

this conversion can help reduce memory usage [28]. The cleaned dataset can be seen in Figure 3.

Table 1. Description of the features in the dataset [24]

S.no	Feature	Description	Type
1	Hash	SHA256 hash value of an Android Application Package (APK) file	Text
2	Millisecond	Unit of time measurement	Number
3	Classification	Malware or clean	Text
4	State	State of processes	Number
5	usage_counter	Usage counter within the task structure	Number
6	prio	Priority of a process (dynamic)	Number
7	Static_prio	Static priority of a process	Number
8	normal_prio	The default priority assigned to a process without considering real-time (RT) inheritance	Number
9	Policy	Process scheduling policies	Number
10	vm_pgoff	Page offset within a virtual memory address	Number
11	vm_truncate_count	A counter that tracks the truncations occurring in a particular memory mapping	Number
12	task_size	Size of the currently executing task	Number
13	cached_hole_size	The size of holes in the virtual address space that are currently not used by a process	Number
14	free_area_cache	Starting address of the hole in the process's virtual address space	Number
15	mm_users	Users utilizing the memory space	Number
16	map_count	Number of memory mappings related to a specific memory region	Number
17	hiwater_rss	High Water Resident Set Size represents the peak of the Resident Set Size (RSS) of a process	Number
18	total_vm	Total number of virtual memory pages	Number
19	shared_vm	Count of shared memory pages	Number
20	exec_vm	Count of executable memory pages	Number
21	reserved_vm	Count of reserved memory pages	Number
22	nr_ptes	Count of page table entries	Number
23	end_data	The end address of the code component present within the Memory layout of a process	Number
24	last_interval	Time duration of the last interval before trashing takes place	Number
25	Nvcsw	Count of voluntary context switches by a process	Number
26	Nivcsw	Count of involuntary context switches by a process	Number
27	min_flt	Number of minor faults that take place in a process	Number
28	maj_flt	Number of major faults that take place in a process	Number
29	fs_excel_counter	File system exclusive resources count	Number
30	Lock	Read-write synchronization mechanism to control the parallel access to the resources of a file system	Number

31	Utime	The CPU time used by a process during the execution of a user-level Code	Number

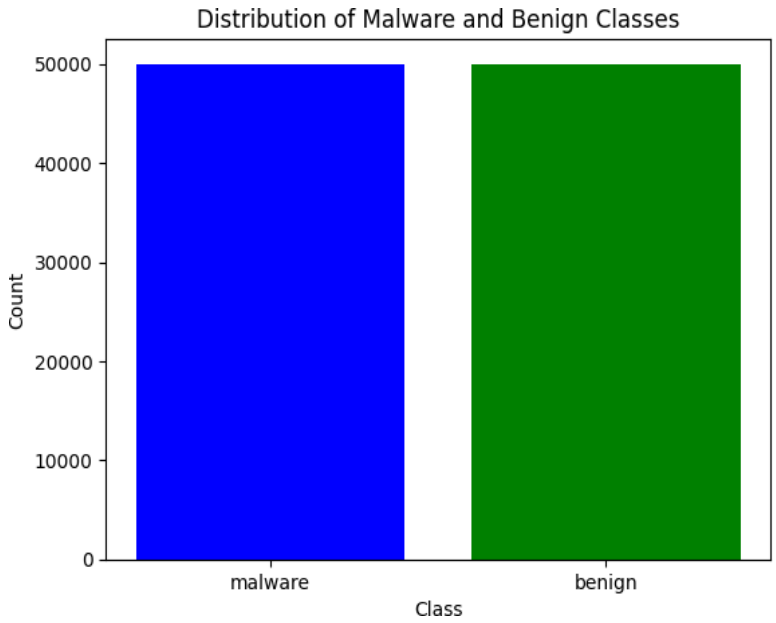

Figure 2. Distribution of malware and benign classes in the dataset

	hash	millisecond	classification	state	usage_cou	prio	static_prio	normal_pi	policy	vm_pgoff	vm_trunc	task_size	cached_h	free_area	mm_users	map_cour	hiwater_r	total_vm	shared_vr
2	42fb5e2ec	0	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
3	42fb5e2ec	1	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
4	42fb5e2ec	2	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
5	42fb5e2ec	3	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
6	42fb5e2ec	4	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
7	42fb5e2ec	5	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
8	42fb5e2ec	6	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
9	42fb5e2ec	7	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
10	42fb5e2ec	8	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	24	724	6850	0	150	120
11	42fb5e2ec	9	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	25	724	6852	0	150	120
12	42fb5e2ec	10	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	25	724	6852	0	150	120
13	42fb5e2ec	11	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	25	724	6852	0	150	120
14	42fb5e2ec	12	1	0	0	3.07E+09	14274	0	0	0	13173	0	0	26	724	6854	0	151	120

Figure 3. Dataset [27]

3.3. Correlation-based Feature Selection

The feature selection process has to be applied to the dataset of newly identified features created through the feature extraction process. Feature selection will further help to identify the essential features and reduce the noise in the data. This process is crucial to increase accuracy, simplify the model, and decrease overfitting [29]. The Supervised feature selection is used for labelled datasets and includes the target variable. The three main Supervised feature selection methods are Filter, Wrapper, and Embedded. In filter methods, statistical measures are used to select the features. Filter methods do not rely on the learning algorithm, have less computational time, and prevent data overfitting [30]. Correlation-based feature selection is a type of filter method. Correlation is the statistical measure of the relationship or the strength and direction of the linear relationship between two variables. The correlation coefficient represents the degree of the coefficient between the two variables, ranging from -1 to +1. When the correlation coefficient value is greater than 0, it is a positive correlation. When it is less than 0, it is a negative correlation, and equal to 0 indicates no relationship between the compared variables. When the correlation is +1, it is a perfect positive correlation, and -1 denotes a perfect negative correlation. Hence, the coefficient values closer to -1 or +1 indicate a stronger correlation. The independent features with high correlation with the target feature (classification) are selected in the Correlation-based coefficient method. These kinds of features have a strong relationship with the target variable, thereby providing meaningful information in predicting the target. They can efficiently grab hold of the crucial aspects of the data that contribute to detecting malware. Independent features highly correlated with themselves are dropped because they give redundant

information, leading to overfitting and unwanted complexity [31]. As the features were highly correlated (because they shared a lot of common subcomponents), Pearson correlation with an r value of more than 0.8 in absolute terms suffices for this purpose (|(r)| > 0.8), will be taken as sufficient. Maintaining model accuracy while ensuring ease of interpretation and standard feature selection practices makes this threshold favourable. These features were removed to eliminate multicollinearity and lower the model's complexity.

Figure 4 depicts the correlation heatmap of the features in the dataset. The red colours indicate a negative correlation, and the blue colours indicate a positive correlation. The heatmap function in seaborn relies on a correlation matrix. The 'corr()' method from pandas obtains the correlation matrix, which uses the Pearson correlation formula to calculate the correlation values.

Each cell in the heatmap shows the correlation coefficient between two features, and different colours indicate the strength and direction of the correlation. A positive correlation (blue) indicates that when one feature increases, the other also tends to increase. A negative correlation (red) depicts that the other tends to decrease as one feature increases. The intensity of the colour shows the magnitude of the correlation coefficient, so darker shades indicate stronger correlations. It helps to determine dependencies between different features in the dataset.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{[\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2]}} \quad (1)$$

Equation 1 depicts the formula for the Pearson correlation coefficient where X and Y are the different features, $\bar{X}$ and $\hat{Y}$ represent the mean values of X and Y, and r is the correlation coefficient. By analyzing the heatmap, some of the independent features that were highly correlated with themselves were dropped; 'vm_truncate_count', 'shared_vm', 'exec_vm', 'nvcsw', 'maj_flt', and 'utime'. For instance, 'shared_vm' has a high correlation of 1 with the 'end_data' feature, and overall, there are many deep, darker, and lighter colours indicating inconsistency. Hence, the 'shared_vm' feature was considered to be dropped due to its high correlation with other independent features.

The Random Forest algorithm is used to calculate the importance of the feature. The calculation is done by measuring the total reduction in the impurity for a specific feature across the decision trees; higher impurity reduction indicates a more important feature [32]. Impurity is the degree of uncertainty in classifying the instances in a node based on their class labels [33]. 'rf.feature_importances_' obtains the feature importance calculated by the Random Forest classifier. Figure 5 depicts ranking important features (descending) that impact the model's behavior in determining the malware. 'static_prio' has the highest value. Some malware creators may try to manipulate the priority of malware processes to get more resource allocation or escape detection processes. Hence, unexpectedly high static priority values for non-important or stranger processes may indicate potential malware. Analyzing 'free_area_cache', detecting unusual memory allocation or deallocation patterns associated with malware is possible.

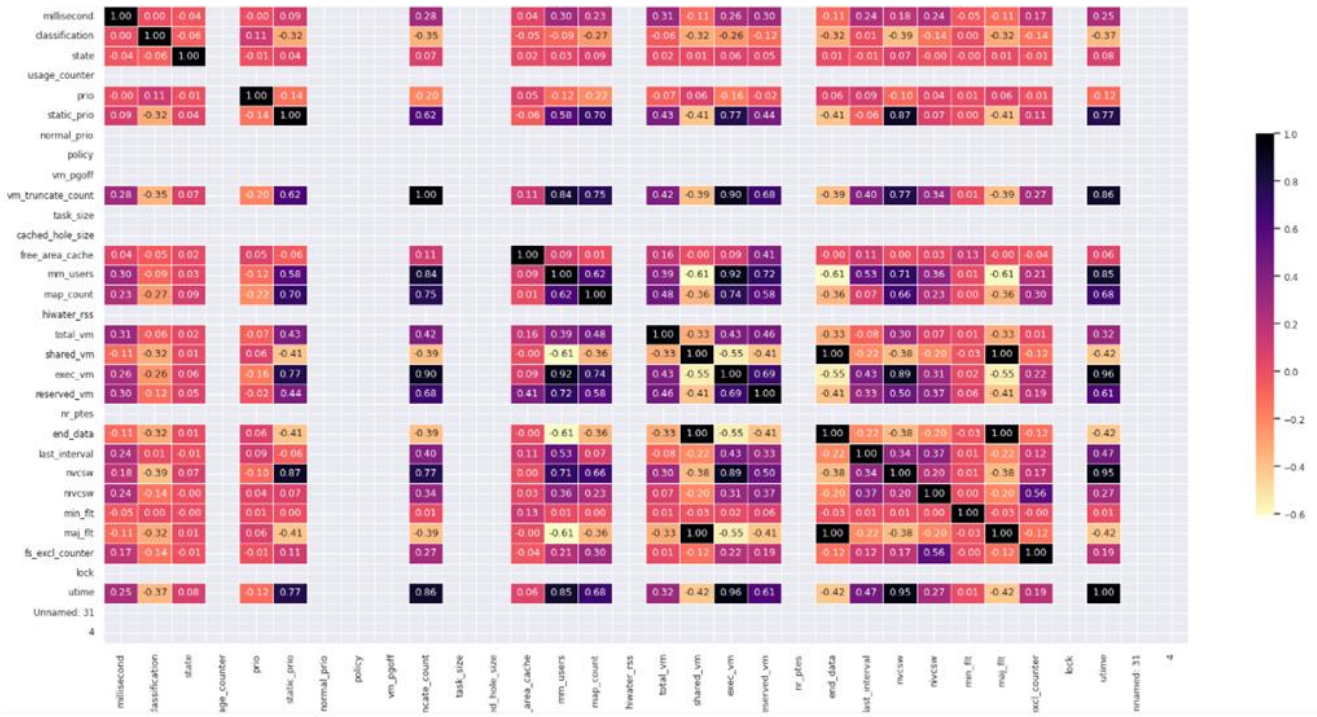

Figure 4. Correlation Heat Map of the features in the dataset

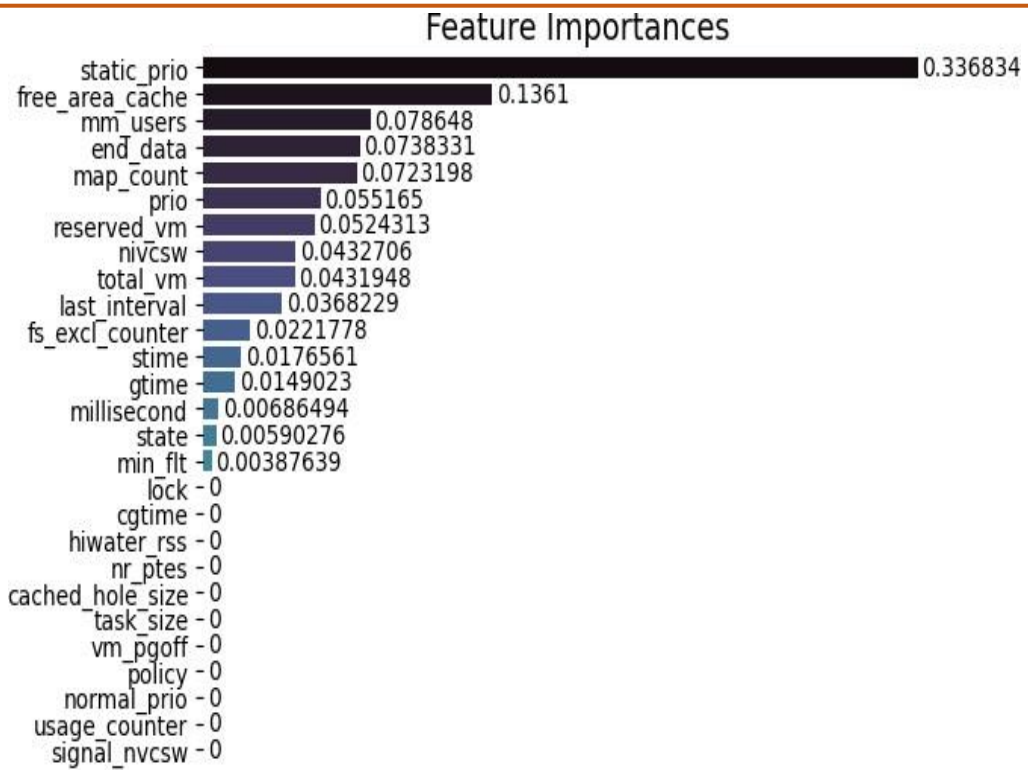

Figure 5. Feature importance

For instance, if a process is continuously allocating and disposing of large memory causing fragmentation, then it may be a clue for malicious processes. 'mm_users' can be used to detect processes with an abnormally high number of users hinting malicious behavior. 'end_data' can be used to observe the processes' memory layout to identify suspicious behavior. 'map_count' can be used to detect abnormalities in memory sharing that can indicate malware. For instance, a process with a very high map_count value compared to other processes can hint at malicious activity [34].

3.4 Classification Process

After selecting the features, the classification process was carried out. Training and testing are the two main stages of the classification process. The data set was divided 28 into a training and a test set. 80% of the dataset was used for training, and 20% was reserved for testing. The algorithm forms a trained model when it is trained with the training set. Then, it is used to classify the test set to determine the model's accuracy. The machine learning algorithms used to train and test the model include Random Forest, Naïve Bayes, Logistic Regression, and Support Vector Machine.

3.4.1 Random Forest

The dataset is split into training and testing sets using the 'train_test_split' from scikit-learn. Each file in the training dataset is associated with a label (malware or clean). Different features are extracted from the files to understand their characteristics; hence, they serve as the input to the algorithm. The 'random_state' parameter is used to ensure reproducibility. The Random Forest object is initialized, and 100 decision trees are set ('n_estimators'). It uses bootstrapping to choose the subsets from the training data randomly and generates several training sets for each decision tree. A random subset of features is used to build the tree, ensuring diversity and avoiding overfitting. Every decision tree is built recursively. The tree at each node selects a feature and threshold to divide the data. The process of splitting increases the separation between the malware and clean files. The features are evaluated and threshold using Gini impurity or information gain. The out-of-bag (OOB, 'oob_score') score is used to get an estimated performance of the model on unseen data during the training phase. The maximum depth ('max_depth') of the decision trees is set to 16. The branches and nodes are created until it reaches the maximum depth. Then, the Random Forest classifier is fitted to the training data. To make predictions, the classifier understands the connections between the features and their corresponding labels. The trained classifier predicts the labels for the test data, and the y_pred variable stores the predicted labels to perform further evaluation. The file that is being classified passes through every tree. Based on the comparison, the file is sent to its appropriate branch. This process occurs until it reaches the leaf node in every tree, where the majority vote of the class label is present [34, 35]. The final result is decided by aggregating the predictions of all the trees through majority voting, where the label with the largest number of votes across the trees is set to the file.

3.4.2 Support Vector Machine (SVM)

Initially, the dataset is split into training and testing sets. An instance of SVM ('kernel') is created with a linear kernel. The goal is to find a hyperplane (decision boundary) that separates malware and clean files with maximum margin. The linear kernel is used since a straight line can separate the data points. The C parameter (set to 1 for balanced regularization) controls the trade-off between increasing the margin and reducing the error. The SVM classifier is trained on the training data. It understands the coefficients related to each feature and the bias term by computing a convex optimization problem. It learns the decision boundary that divides the classes in the data. The trained SVM then makes predictions on the test data. It assigns a prediction to every sample with respect to the decision boundary. The model calculates the distance from the decision boundary for each file to decide which side of the decision boundary the file belongs. The file is clean if the distance is positive and vice versa [36].

3.4.3 Naïve Bayes

Initially, the dataset is split into training and testing sets. An instance of Gaussian Naïve Bayes ('GaussianNB()') is created. This classifier assumes that features follow a normal distribution that is symmetric and bell-shaped, characterized by mean and standard deviation. The classifier is then fit into the training data. It calculates the mean and standard deviation for malware and clean files. The trained classifier then makes predictions on the test data. It calculates the probability of observing the feature values based on the probability distribution of the classes by applying the normal distribution of the calculated mean and standard deviation during the testing phase. The algorithm calculates the probability density function (PDF) using the distribution parameters to determine the probability of a feature value for a particular class. The class prior probability (proportion of instances belonging to a class in the training data) also determines the probability of the data point belonging to a class. Bayes' theorem is applied to combine the probabilities calculated before to get the posterior probability (the probability of a data point associated with a particular class after its feature values are observed) of the classes for the data points. Equation 2 is used to calculate the posterior probability for both classes. The sum of the posterior probability of all the classes equals 1. Zhang [35] pointed out how the Naïve Bayes classifier can reach optimal classification accuracy under specific circumstances, even with its independence assumption simplification.

$$\text{Posterior Probability} = \frac{(\text{Likelihood} * \text{Prior Probability})}{\text{Evidence}} \quad (2)$$

The classifier assigns the class label with the highest posterior probability to the data point. For instance, if the posterior probability of malware is higher than the posterior probability of clean, the data point will be predicted as malware.

3.4.4 Logistic Regression

Initially, the dataset is split into training and testing sets. The logistic regression classifier is initialized ('LogisticRegression(random_state=1)'). The classifier is then fit into the training data. It understands the coefficients of the best-fit logistic function by reducing the error between the predicted and true labels (maximum likelihood or cross-entropy). An iterative algorithm like gradient descent calculates the optimal values for the coefficients. The model provides weights for every input feature. The coefficients indicate the feature's impact on the likelihood of an instance belonging to a positive class. Positive coefficients suggest a positive relationship between the positive class (when the feature value increases, the probability of the instance being classified as malware increases) and vice versa. The coefficient magnitude indicates the strength of the relationship. The trained classifier uses the learned logistic function to predict the labels for the test data. If the predicted probability is above the threshold, the instance is classified as malware and vice versa [36].

4. Analysis and Discussion of Results

The False Positive Rate (FPR) is the proportion of the negative samples that were wrongly classified as positive by the model. It provides information on the ability of the model to detect negative instances accurately. For example, a high FPR can suggest that the model wrongly classifies many clean files as malware [37].

$$FPR = \frac{FP}{(FR+TN)} \quad (3)$$

In Equation 3, FP denotes the false positives, i.e., the number of negative cases that were wrongly classified as positive. TN denotes the true negatives, i.e., the number of negative cases that were correctly identified. True Positive Rate (TPR), also known as recall, is the proportion of samples that were correctly classified as positive. For example, high TPR shows that the model is able to detect malware files effectively while reducing the risk of false negatives.

$$TPR = \frac{TP}{(TP+FN)} \quad (4)$$

In Equation 4, TP denotes true positives, i.e., the number of positive cases that were correctly classified. FN is the false negatives, i.e., the number of positive cases that were wrongly identified as negative. Precision is used to calculate the model's ability to identify positive cases from the total cases that were predicted as positive. High precision indicates low FPs, i.e., the rate

of labeling clean files as malware would be low. Equation 5 shows the formula for calculating precision.

$$Precision = \frac{TP}{(TP+FP)} \quad (5)$$

Accuracy determines the model's overall correctness in detecting the malware. A higher accuracy score indicates that the model has made very few misclassifications. Equation 6 shows the formula for calculating accuracy [37].

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \quad (6)$$

The F1 score combines (harmonic mean) precision and recall, giving a balanced single score. It takes the trade-off between accurately finding malware and reducing FPs. Equation 7 shows the formula for calculating the F1 score.

$$F1\ score = \frac{2*(precision*recall)}{(precision+recall)} \quad (7)$$

Support is the frequency of instances belonging to a class and helps analyze the balance between the classes. Receiver Operator Characteristic (ROC) Graph is plotted with FPR on the x-axis and TPR on the y-axis. When the curve lies above the reference line in the ROC graph, the algorithm performs well, i.e., TPs are higher than FPs and vice-versa. The curves closer to the top-left corner also show a high classifier performance. The ROC curve aids in visually comparing the performance of different models. The Area Under the Curve (AUC) shows the classifier's ability to detect malware, which ranges from 0 to 1, and a higher AUC indicates better model performance. It tells how well the model can discriminate between the malware and clean samples. Macro averaging gives the model's performance across every class unbiasedly. Weighted averaging measures the overall model's performance, taking into account the distribution of the classes.

4.1 Random Forest

Figure 6(a) shows the classification report of Random Forest. TN, FP, FN, and TP are 9795, 176, 169, and 9860, respectively. The precision for the clean class is 0.98, meaning that almost all the instances were classified correctly as clean. Similarly, the model was able to classify most malware correctly. The recall for the clean and malware classes is 0.98, respectively, implying that the model could identify most of the actual clean and malware samples. The F1 score for benign and malware classes is 0.98, suggesting a highly accurate and reliable model. Moreover, Random Forest achieved the highest accuracy of 98%. This validates with the works of Masum *et al.* [15], where a 99.64% accuracy was reported using Random Forest on a large malware dataset, as well as Bawazeer *et al.* [16] who reported high ROC-AUC values with dynamic datasets, confirming our results. The macro average is 0.98, suggesting the model exhibited excellent performance in classifying both classes without bias. Additionally, the weighted average is also 0.98, indicating a high performance considering the distribution of the samples in the dataset.

Figure 6(b) shows the ROC curve of the Random Forest classifier. The curve lies about the reference line and is in the top left corner with an area of 0.99, indicating that Random Forest performs best. The recent work by Zhao *et al.* [17] implementing CNN-LSTM architectures has shown comparable performance, however, Random Forest is still a strong contender due to its reasonable balance of performance, explainability, and speed. The model can effectively differentiate between the classes with high confidence, resulting in highly reliable classification performance. In addition, Shokouhinejad *et al*. [22] explained how Random Forest's interpretability lends itself to ensemble model structures, especially in combination with graph learning and explainability systems, thus supporting its application in practical cybersecurity scenarios. This result matches earlier work by Masum *et al.* [15], who got 99.64% accuracy with Random Forest on a big dataset. Also, Akhtar & Feng [14] reported 92.01% accuracy using RF to classify malware. This high accuracy shows how strong the algorithm is after correlation-based feature selection is used to cut down on noise. Overall, Random Forest showed high performance. Multiple trees reduce the chance of overfitting and increase the model's generalization feature, thereby attaining higher accuracy. The model is also robust to noise and outliers. Bawazeer et al [16] reported similar ROC-AUC values, validating its high discriminative power.

4.2 Support Vector Machine

Figure 7(a) shows the classification report of SVM. The dataset has been split into smaller training and testing sets.TN, FP, FN, and TP are 82, 7, 27, and 84, respectively. Precision for clean is 0.75 and 0.92 for malware class. It also has a recall of 0.92 for clean class, i.e., it catches 92% of true clean files. The recall for the malware class is 76%. The F1 score for the clean class is 0.83, and the malware class is 0.83 indicating a balanced performance in detecting malware. The SVM model demonstrates a good accuracy of 83% in correctly identifying instances belonging to these classes. The SVM model demonstrates strong classification performance with high precision, accuracy, F1 score, and recall values. Figure 7(b) shows the ROC curve of SVM. The proximity of the curve to the upper left corner indicates the SVM model's excellent performance. The model achieves a high TPR while maintaining a low FPR. This is in accordance with what was reported by Akhtar and Feng [14] where 96.01% accuracy was achieved with an SVM and appropriate feature selection. It is possible the accuracy in our study was lower due to feature selection and kernel parameter tuning.

(a)

Class	Precision	Recall	F1-Score	Support
Benign	0.98	0.98	0.98	9971
Malware	0.98	0.98	0.98	10029
Accuracy			0.98	20000
Macro Avg	0.98	0.98	0.98	20000
Weighted Avg	0.98	0.98	0.98	20000

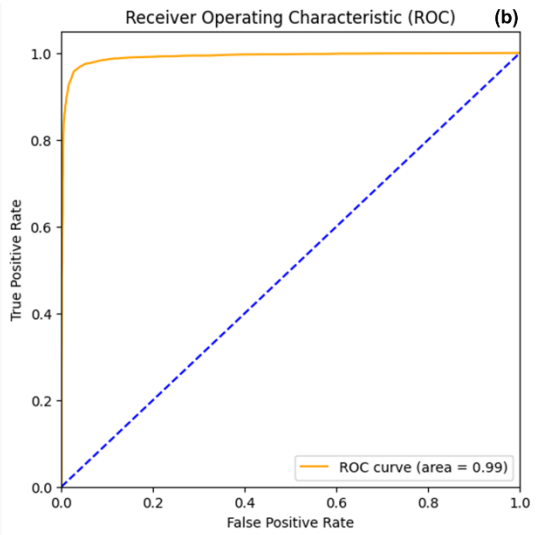

Figure 6(a). Classification Report of Random Forest, **(b).** ROC curve of Random Forest

(a)

Class	Precision	Recall	F1-Score	Support
Benign	0.75	0.92	0.83	10038
Malware	0.92	0.76	0.83	9962
Accuracy			0.83	20000
Macro Avg	0.84	0.84	0.83	20000
Weighted Avg	0.85	0.83	0.83	20000

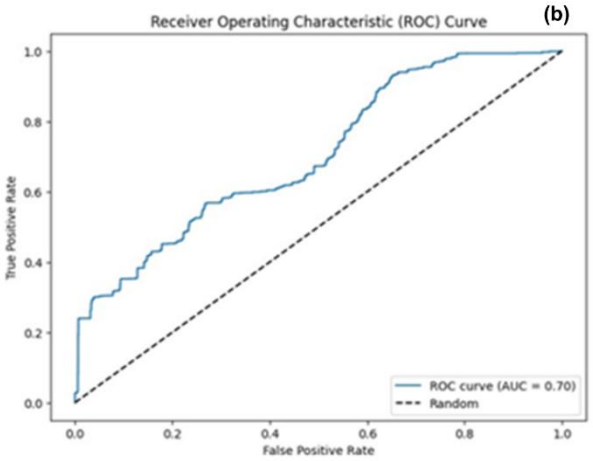

Figure 7 (a). Classification Report of SVM, **(b)** ROC curve of SVM

Shokouhinejad *et al.* [22] noted that for all their classification strengths, SVMs can benefit from graph-based feature representations and explainability methodologies. In other words, the SVM model successfully recognizes a large proportion of positive instances while reducing the number of FPs.

4.3 Naïve Bayes

Figure 8(a) shows the classification report of Naïve Bayes. TN, FP, FN, and TP scores are 3648, 6390, 1020, and 8942, respectively. The precision for clean files is 0.78, which correctly predicts malware class with a precision of 0.58. Moreover, the recall for the clean class is 36%, and it accurately detects the actual malware files with a recall of 90%. The F1 score for the clean class is 0.50 suggesting a moderate performance in recognizing the clean files. However, the F1 score for the malware class is 0.71 is a relatively good performance in accurately recognizing malware files. The model achieved an overall accuracy of 0.63, which indicates that 63% of the instances were correctly classified. Figure 8(b) shows the ROC curve for Naïve Bayes. The AUC value of 0.70 suggests the model has a reasonable balance between sensitivity and specificity, reducing FPs and FNs relatively.

While Akhtar & Feng [14] claimed an accuracy of 89.71% with Naïve Bayes, our model reached 63%. This is most likely due to the Naïve Bayes approach of feature independence, an assumption that does not hold in the context of malware datasets with behavioral indicators that are inter-correlated. Gond and Mohapatra [23] have also reported that these models are not suited to environments where concept drift occurs, such as the dynamic evolution of malware patterns over time. While the model is fast and efficient, its simplicity constrains model effectiveness in modern malware detection and in scenarios where features interact in a non-linear manner. Overall, the Naive Bayes model shows varying performance for different classes, with higher precision, recall, and F1 scores for the malware class than the clean class.

4.4 Logistic Regression

Figure 9(a) shows the classification report of Logistic Regression. TN, FP, FN, and TP scores are 1969, 8069, 1576, and 8,386, respectively. The precision of the benign class is 0.56, and the malware class is 0.51, implying that 51% of the instances predicted as malware were truly malware. The recall for the clean class is just 0.20, while for the malware class is 0.84, thereby accurately identifying 84% of the true malware. The F1 score for the benign class is 0.29, while for the malware class is 0.63, showing a better balance between precision and recall. Figure 9(b) shows the ROC curve of Logistic Regression. The AUC is 0.58, which suggests that the classifier performs relatively low in discriminating the malware and benign classes compared to the other studies [15] and the theoretical optimality of Naïve Bayes classification demonstrated in Zhang [35], which explains why even simple probabilistic models can yield competitive results under certain conditions. This lack of performance aligns with Alsumaidaee et al. [24], who noted that linear models such as logistic regression do not effectively capture intricate behavioural patterns within malware data. Although the model is easy to interpret and computationally efficient, it is unable to capture non-linear relationships between features that frequently occur in malware detection problems. More recent works advocate hybrid and deep learning approaches over linear classifiers for such tasks, particularly in highly dynamic or imbalanced datasets. Overall, the model struggles to detect benign files. However, it performs comparatively better in recognizing malware samples, exhibiting a modest performance.

The linear structures of Naïve Bayes and Logistic Regression allowed them to train markedly faster, from a computational efficiency perspective. SVM took longer to train, particularly with large datasets and non-linear kernels. Random Forest did improve accuracy; however, the ensemble of multiple decision trees increased the computational cost, which is critical for real-time or resource-limited systems.

(a)

Class	Precision	Recall	F1-Score	Support
Benign	0.78	0.36	0.50	10038
Malware	0.58	0.90	0.71	9962
Accuracy			0.63	20000
Macro Avg	0.68	0.63	0.60	20000
Weighted Avg	0.68	0.63	0.60	20000

(b)

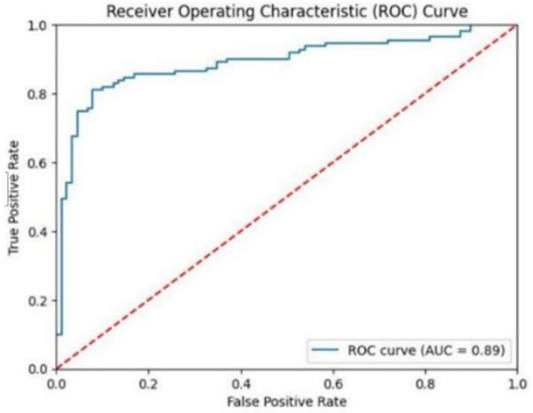

Figure 8(a). Classification Report of Naïve Bayes, **(b).** ROC curve of Naïve Bayes

(a)

Class	Precision	Recall	F1-Score	Support
Benign	0.56	0.20	0.29	10038
Malware	0.51	0.84	0.63	9962
Accuracy			0.52	20000
Macro Avg	0.53	0.52	0.46	20000
Weighted Avg	0.53	0.52	0.46	20000

(b)

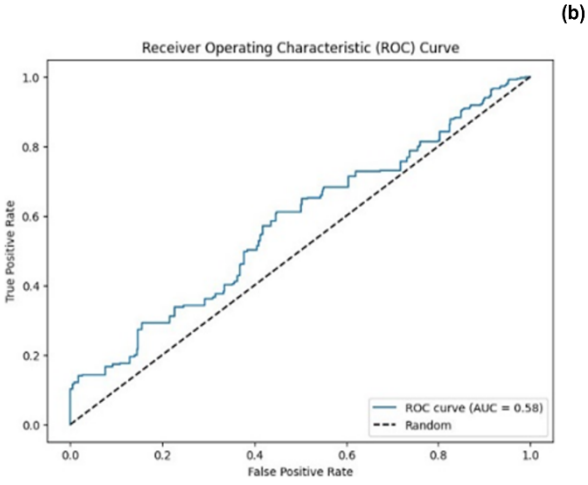

Figure 9(a). Classification Report of Logistic Regression, **(b).**ROC curve of Logistic Regression

4.5 Performance comparison of the proposed classifiers

An ablation study evaluated correlation-based feature selection by comparing the performance of all classifiers, including those with and without feature selection. Table 2 shows the result summary and comparison between different classifiers. It is observed that feature selection improved remarkable accuracy up to 99% in RF classifiers. Similarly, the feature selection increased in SVM, Naïve Bayes, and Logistic regression. The improvement in accuracy can be attributed to the removal of redundant and highly correlated features, which reduced noise and retaining features which is correlated to the target class. RF and SVM are performing well compared to the other proposed algorithms.

The Random Forest classifier achieved the best results with an accuracy of 99%, with SVM following at 94%, Naïve Bayes at 63%, and Logistic Regression only at 50%. Random Forest's performance is explained by the ensemble learning technique, which mitigates overfitting by constructing multiple decision trees, and by its capacity to model complex and non-linear system-behavior relationships, consistent with findings by Masum *et al*. [15] and Bawazeer *et al*. [16]. SVM performance was also strong because it identifies optimal hyperplanes that separate malware from benign classes, but its accuracy is lower, possibly due to feature scaling and dataset size, which is consistent with findings of Akhtar & Feng [14]. Naïve Bayes and Logistic Regression likely underperformed because the strong independence assumption of Naïve Bayes fails with malware datasets that contain correlated features, and the linear decision boundary of Logistic Regression is inadequate for describing the non-linear relationships [38]. The analysis shows that feature selection, paired with ensemble or kernel-based models, meets the accuracy, robustness, and generalization limits of malware detection identified in previous studies.

4.6 k-Fold Cross-Validation

Each classifier was evaluated using 10-fold cross-validation to determine its generalization ability and check for overfitting. The Random Forest classifier continued to obtain an average accuracy of 98.7% with a standard deviation of 0.4% across all folds. This shows that Random Forest obtained consistent results on every fold. The SVM scored 93.8% in accuracy while Naïve Bayes and Logistic Regression recorded lower results, but all performed reliably. Random Forest stands out in performance compared to the other classifiers, achieving accuracy that is not likely due to overfitting, as confirmed through paired t-tests.10 Fold cross-validation result is given in the Table 3.

5. Conclusion

Cybercriminals create malicious software or malware to attack and exploit systems, devices and networks. These types of attacks are on the rise, with malware severely impacting the global economy through lost revenue, stolen data, damages to customer trust, and even business closures. The emergence of more sophisticated types of malware has led to the decline of traditional signature-based detection techniques. Machine learning techniques to detect malware is within the realm of the possible and would alleviate the detection challenges posed by older methods.

Table 2 Performance comparison of RF, SVM, Naïve Bayes, and LR with and without feature selection

Algorithm	Accuracy (No FS)	Accuracy (With FS)	F1-Score (No FS)	F1-Score (With FS)
Random Forest	95%	99%	0.95	0.99
Support Vector Machine	88%	94%	0.88	0.94
Naïve Bayes	60%	63%	0.60	0.63
Logistic Regression	48%	50%	0.48	0.50

Table 3. 10-Fold Cross-Validation Results

Algorithm	Mean Accuracy (%)	Standard Deviation (%)
Random Forest	98.7	0.4
Support Vector Machine	93.8	0.6
Naïve Bayes	62.5	1.2
Logistic Regression	49.8	1.0

This study's methodology centers on the implementation and evaluation of several ML techniques for malware detection and the assessment of these techniques through various scales including accuracy, precision, recall, F1 score, and ROC curves. A data set was encoded, then underwent correlation-based feature selection, and was subsequently divided into training and testing sets. The training and testing phases were conducted with Random Forest, Naïve Bayes, Logistic Regression, and SVM algorithms. The Random Forest model was the most robust, achieving a 99% accuracy rating. The SVM model was also strong, reaching 83%. while Naïve Bayes and Logistic Regression showed modest accuracies of 63% and 52%. Generalizability can be enhanced by including additional diverse samples in the dataset. In modern science, Federated Learning can be applied for malware detection with the support of ML models without sharing raw data, giving more privacy and detection accuracy. So, it is possible to extend this work with the support of federated learning to adapt quickly to the new threats. In image-based malware detection, there is a possibility of converting malware binaries or opcode sequences into grayscale or RGB images. Deep learning, such as CNN, Vision Transformers, and hybrid algorithms like CNN-RNNs, can be applied for classification. So, Future studies might focus on detecting advanced malware and abnormal behavior with ensembles, deep learning, and behavioral analysis. Issues, including data imbalance, concept drift, and adversarial strikes, need attention too. Defenses can be improved with adversarial training and strong feature selection. With the support of reliability theory, it is possible to have enhanced trouble-free software for any cyber-physical systems. In this AI world, many software applications exist in various crucial fields, including the health sector for disease detection. If there is any virus or malfunction in the software, it will create a false prediction of disease, which may result in human loss. So, reliable software is a “must-needed” in various sectors. So, this paper provides a solution to enhance the strength of cybersecurity systems through advanced malware detection to ensure better reliability in software-based systems.

References

[1] M. Sikorski, A. Honig. (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. No starch press.

[2] D. Ucci, L. Aniello, R. Baldoni, A taxonomy of malware behavior for classification and detection. Journal of Computer Virology and Hacking Techniques, 15(2), (2019) 67–90.

[3] L. Coventry, D. Branley, and T. Chesney, Cybersecurity threats and the human factor: Investigating the moderating effect of personal values on cybersecurity awareness. Computers & Security, 105, (2021) 102247.

[4] M. Conti, A. Dehghantanha, K. Franke, S. Watson, Internet of things security and forensics: Challenges and opportunities. Future Generation Computer Systems, 78, (2018) 544–546. https://doi.org/10.1016/j.future.2017.07.060

[5] R. Amuthakkannan, S.M. Kannan, K. Vijayalakshmi, N. Ramaraj, Reliability Analysis of Programmable Mechatronics System Using Bayesian Approach. International Journal of Industrial and Systems Engineering, 4(3), (2009) 303-325. https://doi.org/10.1504/IJISE.2009.023544

[6] K. Vijayalakshmi, Reliability improvement in component-based software development environment. International Journal of Information Systems and Change Management, 5(2), (2011) 99–123. https://doi.org/10.1504/IJISCM.2011.041510

[7] T. Fox, Cybercrime to cost the world $12.2 trillion annually by 2031, Cybersecurity Ventures – 2025 Official Cybercrime Report (Blog), May 28, 2025. [Online]. Available: https://cybersecurityventures.com/official-cybercrime-report-2025/

[8] M. F. Safitra, M. Lubis, and H. Fakhrurroja, Counterattacking cyber threats: A framework for the future of cybersecurity, Sustainability, 15 (2023) 13369. https://doi.org/10.3390/su151813369

[9] A.L. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), (2016) 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

[10] A. Bensaoud, J. Kalita, M. Bensaoud, A survey of malware detection using deep learning. Machine Learning with Applications, 16, (2024) 100546. https://doi.org/10.1016/j.mlwa.2024.100546

[11] A. Pinhero, M.L. Anupama, P. Vinod, C.A. Visaggio, N. Aneesh, S. Abhijith, S. AnanthaKumar, Malware detection employed by visualization and deep neural network. Computers & Security, 105, (2021) 102247. https://doi.org/10.1016/j.cose.2021.102247

[12] A. Hussain, A. Saadia, A. Gul, M. Alhussein, K. Aurangzeb, Enhancing ransomware defense: Deep learning-based detection and family-wise classification of evolving threats. PeerJ Computer Science, 10, (2024) e2546. https://doi.org/10.7717/peerj-cs.2546

[13] H.A.K. Harsha, T.A. Murthy, Machine-learning techniques for malware detection. International Journal of Scientific Research in Science, Engineering and Technology, 8(5), (2021) 70–76. https://doi.org/10.32628/IJSRSET21858

[14] R. Amuthakkannan, Parameters Design and Performance Analysis of a Software-Based Mechatronics System Using Taguchi Robust

Design – A Case Study. International Journal of Productivity and Quality Management. 10(1), (2012) 1–24. https://dx.doi.org/10.1504/IJPQM.2012.047939

[15] K. Vijayalakshmi, N. Ramaraj, R. Amuthakkannan, Improvement of component selection process using genetic algorithm for component-based software development. International Journal of Information Systems and Change Management, 3(1), (2008) 63–80. https://dx.doi.org/10.1504/IJISCM.2008.019289

[16] [M.S. Akhtar, T. Feng, Malware analysis and detection using machine-learning algorithms. Symmetry, 14(11), (2022) 2304. https://doi.org/10.3390/sym14112304

[17] M. Masum, M.J.H. Faruk, H. Shahriar, K. Qian, D. Lo, M.I. Adnan, Ransomware classification and detection with machine learning algorithms. In 2022 IEEE 12th annual computing and communication workshop and conference (CCWC), IEEE, Las Vegas, NV, USA, 0316-0322.

[18] O. Bawazeer, T. Helmy, S. Al-Hadhrami, Malware detection using machine learning algorithms based on hardware performance counters: Analysis and simulation. Journal of Physics: Conference Seriesvol. 1962(1), (2021) 012010. https://doi.org/10.1088/1742-6596/1962/1/012010

[19] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in Proc. 2016 IEEE 27th Int. Symp. on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 2016, pp. 309–320. https://doi.org/10.1109/ISSRE.2016.13

[20] M. Alshomrani, A. Albeshri, A. A. Alsulami, and B. Alturki, An explainable hybrid CNN–Transformer architecture for visual malware classification, Sensors, 25(15), (2025) 4581. https://doi.org/10.3390/s25154581

[21] M. Alshamrani, S. Alshammari, M. Alasmary, Federated learning for malware detection in IoT edge networks. Future Generation Computer Systems, 146, (2024) 109–120. https://doi.org/10.1016/j.comnet.2021.108693

[22] H. Shokouhinejad, R. Razavi-Far, H. Mohammadian, M. Rabbani, S. Ansong, G. Higgins, A.A. Ghorbani, (2025) recent advances in malware detection: Graph learning and explainability. arXiv preprint arXiv:2502.10556. https://doi.org/10.48550/arXiv.2502.10556

[23] B.P. Gond, D.P. Mohapatra, (2025) deep learning-driven malware classification with API call sequence analysis and concept drift handling. arXiv preprint arXiv:2502.08679. https://doi.org/10.48550/arXiv.2502.08679

[24] Y.A.M. Alsumaidaee, M.M. Yahya, A.H. Yaseen, Optimizing malware detection and classification in real-time using hybrid deep learning approaches. International Journal of Safety and Security Engineering, 15(1), (2025) 15–25. https://doi.org/10.18280/ijsse.150115

[25] D.M. Powers, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. International Journal of Machine Learning Technology, 2(1), (2011) 37–63. https://doi.org/10.48550/arXiv.2010.16061

[26] Q. Abu Al-Haija, A. Odeh, H. Qattous, PDF malware detection based on optimizable decision trees. Electronics, 11(19), (2022) 3142. https://doi.org/10.3390/electronics11193142

[27] N. Saravana, (2018) Malware Detection | Kaggle, [Online]. Available: https://www.kaggle.com/datasets/nsaravana/malware-detection?select=Malware+dataset.csv

[28] J. T. Hancock and T. M. Khoshgoftaar, Survey on categorical data for neural networks, Journal of Big Data, 7(1) (2020) 28. https://doi.org/10.1186/s40537-020-00305-w

[29] G. Chandrashekar, F. Sahin, A survey on feature selection methods. Comput. Electr. Eng., 40(1), (2014) 16–28. https://doi.org/10.1016/j.compeleceng.2013.11.024

[30] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, H. Liu, Feature selection: A data perspective. ACM computing surveys (CSUR), 50(6), (2017) 1-45. https://doi.org/10.1145/3136625

[31] M.A. Hall. (2000). Correlation-based feature selection for discrete and numeric class machine learning. In Proceeding University of Waikato.

[32] M. Belgiu, L. Drăguț, Random forest in remote sensing: A review of applications and future directions, ISPRS J. Photogramm. Remote Sens., 114, (2016) 24–31. https://doi.org/10.1016/j.isprsjprs.2016.01.011

[33] G. Biau, E. Scornet, A random forest guided tour. Test, 25(2), (2016) 197–227. https://doi.org/10.1007/s11749-016-0481-7

[34] C. Cortes, V. Vapnik, Support-vector networks. Maching Learning, 20(3), (1995) 273–297. https://doi.org/10.1007/BF00994018

[35] H. Zhang. (2004). the optimality of naïve Bayes. in Proc. 17th Int. Florida Artificial Intelligence Research Society Conf. (FLAIRS), Miami Beach, FL, USA, 562–567.

[36] C.Y.J. Peng, K.L. Lee, G.M. Ingersoll, An introduction to logistic regression analysis and reporting. The journal of educational research, 96(1), (2002) 3–14. https://doi.org/10.1080/00220670209598786

[37] J. Ding, V. Tarokh, Y. Yang, Model selection

techniques: An overview. IEEE Signal Processing Magazine, 35(6), (2018) 16–34. https://doi.org/10.1109/MSP.2018.2867638

[38] Rakibul Hasan, Barna Biswas, Md Samiun, Mohammad Abu Saleh, Mani Prabha, Jahanara Akter, Fatema Haque Joya & Masuk Abdullah, Enhancing malware detection with feature selection and scaling techniques using machine learning models, Scientific Reports, 15 (2025) 9122. https://doi.org/10.1038/s41598-025-93447-x

Authors Contribution Statement

K. Vijayalakshmi: Supervision, Writing - Review & Editing. Eshaa Balamurugan Subhashini: Methodology, Formal analysis, Data curation, Software, Writing - Original Draft. Safiya Al Sabahi: Methodology, Formal analysis, Software. Rahmah Al Shanawi: Formal analysis, Writing - Original Draft. Tahani Al Harthy: Data Curation, Formal Analysis, Writing - Original Draft. All authors have read and approved the final version of the manuscript.

Funding

The authors declare that no funds, grants or any other support were received during the preparation of this manuscript.

Competing Interests

The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Data Availability

The data supporting the findings of this study can be obtained from the corresponding author upon reasonable request.

Has this article screened for similarity?

Yes