2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN) | 979-8-3315-3038-9/25/$31.00 ©2025 IEEE | DOI: 10.1109/CICTN64563.2025.10932606

Behavior-Based Machine Learning Approaches for Malware Detection

Vivek Srivastava
Department of Computer Science and Engineering
ABES Engineering College, Ghaziabad
Email: viveksrivastava@abes.ac.in

Manya Khare
Department of Computer Science and Engineering
ABES Engineering College, Ghaziabad
Email: manya.21b0131054@abes.ac.in

Manasvi Kansal
Department of Computer Science and Engineering
ABES Engineering College, Ghaziabad
Email: manasvi.21b0101013@abes.ac.in

Harsh
Department of Computer Science and Engineering
ABES Engineering College, Ghaziabad
Email: harsh.21b0101050@abes.ac.in

***Abstract*—The present research investigates the implementation of various methods of machine learning to analyze fluid information from executable files for the purpose identify contamination. We extracted feature vectors from a dataset that contained both malicious and benign samples. Standard metrics like F1-score, recall, and precision were used to train and evaluate a range of classifiers, including Decision Trees, Random Forests, Support Vector Machines, and Logistic Regression. Our test findings show that the models successfully distinguish between harmful and benign files, with Random Forest providing the highest level of consistency in all evaluation measures. These Results demonstrate the promise of machine learning methods to improve systems for detecting malware.**

***Index Terms*—Classification, Decision Trees, Logistic Regression, Machine Learning, Malware Detection, Random Forest, Support Vector Machines**

I. Introduction

With the exponential growth of digital technologies, the threat landscape for cyberattacks has widened significantly [1]. One particularly dangerous form of attack is malware, which can infiltrate systems, steal sensitive data, and disrupt operations [2]. Identifying and mitigating malware threats is crucial for ensuring system security and integrity [3]. Over the years, machine learning techniques have gained significant traction in this domain, offering robust methods for detecting and classifying malware based on behavioral patterns and data analysis [4]. This research investigation examines malware detection using a variety of machine learning models, including Random Forests, Decision Trees, Support Vector Classifiers, and Logistic Regression. [5]. An effective intrusion detection framework based on SVM with feature augmentation has been shown to enhance detection accuracy and robustness [6]. In an effort to assess these algorithms' effectiveness and obtain high classification accuracy, they were trained using a dynamic malware dataset. To enhance detection performance, a hybrid network intrusion detection framework that combines weighted k-means and random forests has been created. [7]. To enhance training effectiveness and performance, a novel feedforward neural network learning technique called the extreme learning machine had been suggested. [8]. Our research applies standard preprocessing steps, such as feature extraction and scaling, followed by model training and evaluation. A comparison of each algorithm's efficacy is made using criteria such as recall, precision, F1-score, and total accuracy. The knowledge gathered from this study helps to the expanding field of automated malware detection, emphasizing how crucial it is to use sophisticated algorithms to changing cyber-threats.

The insights gained from this research contribute to the growing field of automated malware detection, highlighting the importance of using advanced algorithms to counter evolving cyber threats.

II. Related Work

Various machine learning techniques, including SVM, Decision Trees, and deep learning models like CNN and LSTM, have been applied to malware detection. These methods offer promising results in detecting both known and unknown malware variants. Previous research has demonstrated the effectiveness of dimensionality reduction techniques like PCA in handling large datasets. Owing to the escalating threat posed by rogue applications Android detection of ransomware has attracted a lot of attention over the past decade. Various approaches have emerged to combat this issue, each leveraging distinct methodologies and datasets. As software complexity and security concerns increase, malware has emerged as a serious threat to computer systems. This chapter talks about machine learning's application in malware detection and investigation important defenses against malware attacks, emphasizing current malware threat trends [9]. Almomani et al. (2021) proposed a hybrid evolutionary approach that efficiently addresses the challenges posed by highly imbalanced datasets, thus enhancing the detection accuracy of ransomware variants [10]. With the objective to boost the accuracy of detection through more precise feature selection, Bibi et al. (2019) proposed a multi-factor feature filtration strategy in conjunction with recurrent neural networks [11]. The landscape of deep learning methods used for ransomware detection was compiled in a review by Alzahrani and Alghazzawi (2019), which highlighted the models' capacity to recognize complex threats [12]. Meanwhile, Sharma et al. (2021) utilized unsupervised machine learning techniques for forensic analysis, highlighting the importance of understanding ransomware behavior [13]. Qaddoura et al. (2021) demonstrated that evolutionary clustering methods could enhance classification accuracy in ransomware detection, while Sheen and Gayathri (2022) focused on early detection through foreground activity analysis [14], [15]. Furthermore, Manavi and Hamzeh (2022) exhibited innovative techniques in the field by suggesting a novel detection method based on PE headers that uses graph embedding [16]. Abdullah et al. (2020) emphasized the significance of dynamic feature extraction in their approach, which was further complemented by Alsoghyer and Almomani (2020), who analyzed application permissions to identify potential vulnerabilities [17], [18]. On a broader scale, Zadeh Nojoo Kambar et al. (2022) and Masum et al. (2022) contributed surveys and classifications of various mobile malware detection methods, highlighting the importance of machine learning techniques in enhancing cybersecurity measures against ransomware and other malicious software [19], [20]. Overall, these studies reflect a concerted effort within the research community to develop more effective strategies for ransomware detection and prevention.

III. System Architecture

The proposed system is designed to detect malware based on behavioral features extracted from the execution of both malicious and benign software samples.

A. Data Collection

System calls, registry modifications, file actions, and network activity are examples of dynamic malware behavior data found in the dataset utilized in this investigation. This data was collected from malware analysis sandboxes and other real-world sources, providing a diverse set of features for classification tasks.

The dataset includes N = 34, 371 examples, each of which has a label indicating whether the behavior is malevolent or benign and is described by F = 500 behavioral factors.

Assume Z is a member of R. The feature matrix is represented by $\mathbf{Z} \in \mathbb{R}^{N \times F}$, where each row, $\mathbf{z}_i = \{z_{i1}, z_{i2}, \ldots, z_{iF}\}$, symbolizes the feature vector of the i-th sample. The symbol $\mathbf{l} \in \{-1, 1\}^N$ represents the label vector, where the classification is specified for each element.

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1F} \\ z_{21} & z_{22} & \cdots & z_{2F} \\ \vdots & \vdots & \ddots & \vdots \\ z_{N1} & z_{N2} & \cdots & z_{NF} \end{bmatrix}, \quad \mathbf{l} = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_N \end{bmatrix}$$

At this point the j-th feature of the i-th sample is indicated by z_{ij}, and the label is indicated by $l_i \in \{-1, 1\}$, where $l_i = -1$ represents benign behavior and $l_i = 1$ indicates malicious conduct. In order to assess the model's performance,

TABLE I
Comparison of Ransomware Detection Approaches

Reference	Detection Technique	Dataset Used	ML Method	Key Contribution
[10]	Hybrid Evolutionary Approach	Highly imbalanced dataset	Evolutionary Clustering	Handles imbalanced data efficiently
[11]	Multi-factor Feature Filtration	Custom dataset	RNN	Combines feature filtration and RNN for enhanced detection
[13]	Unsupervised ML	Dynamic ransomware samples	Unsupervised Learning	Forensic analysis of Android ransomware
[14]	Evolutionary Clustering	Custom dataset	Evolutionary Clustering Algorithm	Uses clustering for enhanced classification accuracy
[15]	Foreground Activity Analysis	Locker ransomware dataset	Supervised Learning	Early detection of Android Locker ransomware
[16]	PE Header Analysis with Graph Embedding	PE header dataset	GNN	Novel approach using graph embedding for ransomware detection
[17]	Dynamic Feature Extraction	Dynamically obtained features	Classification Algorithms	Focus on dynamic feature extraction for Android ransomware
[18]	Application Permission Analysis	Application permissions dataset	ML	Focuses on effectiveness of app permissions for ransomware detection
[19]	Mobile Malware Detection	Various datasets	ML	Survey on different methods for mobile malware detection
[20]	Ransomware Classification	Custom dataset	ML and Neural Networks	Combines classification and feature selection for improved detection

the dataset is split into training and testing subsets in an 80-20 ratio: (Ztrain, ltrain), (Ztest, ltest) This partition data provides a reliable assessment of the model's prediction ability by testing it on unknown data.

B. Feature Extraction

In this work, we extract multiple behavioral features from each malware sample, such as API calls, file system operations, and network activities. Let $\mathbf{V} = \{v_1, v_2, \ldots, v_q\}$ denote the set of extracted features from a malware sample, wherein the aggregate amount of features has been indicated by q. These characteristics have been divided specifically into three groups:

- **API Calls**: Let $\mathbf{A} = \{a_1, a_2, \ldots, a_r\}$ represent the set of API calls, where r is the total number of distinct API calls observed.
- **File System Operations**: Let $\mathbf{F} = \{f_1, f_2, \ldots, f_s\}$ denote the set of file system operations, where s indicates the number of observed file-related activities.
- **Network Activities**: Let $\mathbf{N} = \{n_1, n_2, \ldots, n_t\}$ represent the set of network interactions, where t corresponds to the total number of network activities.

These features are encoded as a feature vector $\mathbf{V} = [v_1, v_2, \ldots, v_q]$, which serves as the input to our classification model. The goal of the model is to map the feature vector $\mathbf{V}$ to a label $l \in \{-1, 1\}$, where $l = -1$ indicates benign behavior, and $l = 1$ represents malicious behavior.

C. Data Preprocessing

To mitigate the immense dimensionality of the dataset, researchers employ Principal Component Analysis (PCA). PCA shows the information on a subspace with less dimensions with regard to the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of samples and d is the number of features. The projection is carried out by determining which k main components capture the most variance, such that X becomes a reduced matrix $\mathbf{X}' \in R$. N × k, where k is greater than d. This is expressed mathematically as:

$$\mathbf{X}' = \mathbf{X}\mathbf{W},$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is the matrix of the top k eigenvectors corresponding to the largest eigenvalues of the covariance matrix of $\mathbf{X}$.

Additionally, the dataset is often imbalanced, with more benign samples than malicious ones. Researchers utilize the Synthetic Minority Over-sampling Technique (SMOTE) to address this. Given the minority class samples $M = \{m_1, m_2, \ldots, m_q\}$, SMOTE generates synthetic samples by interpolating between m_i and its k-nearest neighbors. The new synthetic samples are added to balance the dataset, ensuring that the number of minority samples matches the majority class, before training the model.

D. Model Training

We employ several machine learning models, including:

- **Support Vector Classifier** Support Vector Classifiers (SVC) are effective methods for binary classification. They establish a hyperplane that optimally distinguishes two classes within the dataset. Given training samples $(\mathbf{z}_j, c_j)$, where $\mathbf{z}_j \in \mathbb{R}^d$ are feature vectors and $c_j \in \{-1, 1\}$ are class labels, the SVC aims to maximize the separation margin for ideal classification.
  Hard-Margin SVC:
  The intent is to maximize the margin when the data is linearly separable by addressing:
  $$\min_{\mathbf{u},d} \frac{1}{2}\|\mathbf{u}\|^2 \tag{1}$$
  subject to:
  $$c_j(\mathbf{u}^T\mathbf{z}_j + d) \geq 1 \quad \forall j. \tag{2}$$
  Here, $\mathbf{u}$ denotes the normal vector to the hyperplane, and d is the bias term. Minimizing $\|\mathbf{u}\|^2$ corresponds to maximizing the margin, ensuring that all points are correctly classified.
  Soft-Margin SVC:
  In practical situations, when the data is not perfectly separable, slack variables η_j are introduced to permit a certain degree of misclassification. The optimization problem is then defined as:
  $$\min_{\mathbf{u},d,\eta} \frac{1}{2}\|\mathbf{u}\|^2 + C\sum_{j=1}^{n}\eta_j \tag{3}$$
  subject to:
  $$c_j(\mathbf{u}^T\mathbf{z}_j + d) \geq 1 - \eta_j, \quad \eta_j \geq 0 \quad \forall j. \tag{4}$$
  On this scenario, the trade-off between boosting the margin and minimizing classification lapses is regulated by the regularization value C. The slack variables η_j allow the margin constraint to be broken at some places, increasing the data's robustness to noise.
  Kernel SVC:
  SVC can be stretched to project data into a higher-dimensional space by utilizing kernel functions when the data is not linearly separable in the original feature space. The formulation of the dual optimization problem is as:
  $$\min_{\beta} \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n}\beta_j\beta_k c_j c_k K(\mathbf{z}_j, \mathbf{z}_k) - \sum_{j=1}^{n}\beta_j \tag{5}$$
  subject to:
  $$\sum_{j=1}^{n}\beta_j c_j = 0, \quad 0 \leq \beta_j \leq C. \tag{6}$$
  In this case, $K(\mathbf{z}_j, \mathbf{z}_k)$ is the kernel function that computes the inner product in the transformed space, allowing SVC to discover non-linear decision boundaries. Commonly used kernels include linear, polynomial, and radial basis function (RBF). The support vectors, identified by the non-zero Lagrange multipliers β_j, are the data points that define the decision boundary.
- **Decision Tree:** Non-parametric supervised learning methods called decision trees are used for tasks involving

regression and classification. A decision tree's main goal is to divide the input space into regions, each of which is connected to having a certain value or class designation.

Tree Construction:

At every internal node of the tree, the dataset is divided based on a feature that optimizes a particular criterion, commonly Gini impurity or entropy. For a given node p, the objective is to minimize the impurity $J(p)$.

Gini Impurity:

$$G(p) = 1 - \sum_{j=1}^{K} q_j^2 \tag{7}$$

where q_j represents the proportion of samples belonging to class j at node p, and K is the total number of classes. The goal is to choose the feature that minimizes the weighted Gini impurity across the child nodes.

Entropy:

$$E(p) = -\sum_{j=1}^{K} q_j \log(q_j) \tag{8}$$

Entropy quantifies the degree of uncertainty or disorder at a node, and the optimal feature is the one that minimizes the entropy following the split.

Information Gain:

For a feature y_k, the information gain is calculated as:

$$\Delta E = E(p) - \sum_{m} \frac{|p_m|}{|p|} E(p_m) \tag{9}$$

where p_m are the child nodes after the split, and $|p|$ denotes the number of samples at node p.

Recursive Partitioning:

The tree is constructed recursively by selecting the most effective feature y_k at each node based on maximizing information gain. This procedure keeps on until a stopping criterion—like a minimum number of samples within a node or a maximum tree depth—is met.

Pruning:

To prevent overfitting, post-pruning techniques, such as cost-complexity pruning, are utilized. Here, the complexity β of the tree is penalized:

$$\text{Cost}(R) = \sum_{p \in R} |p| J(p) + \beta |R| \tag{10}$$

where R represents the set of leaf nodes in the tree.

- **Logistic Regression:** Logistic Regression is a popular classification algorithm that estimates the probability of a binary outcome. Given a vector of input features $\mathbf{v} = (v_1, v_2, \ldots, v_m) \in \mathbb{R}^m$ and a corresponding label $c \in \{0, 1\}$, the model predicts the probability of $c = 1$ using a sigmoid function:

$$P(c = 1 \mid \mathbf{v}) = \sigma(\mathbf{u}^T \mathbf{v} + d), \tag{11}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic sigmoid function, $\mathbf{u} \in \mathbb{R}^m$ is the parameter vector, and d is the bias term. The model is trained by minimizing the binary cross-entropy loss defined as:

$$\begin{aligned} L(\mathbf{u}, d) = -\frac{1}{n} \sum_{k=1}^{n} \Big[& c^{(k)} \log(h_{\mathbf{u}}(\mathbf{v}^{(k)})) \\ & + (1 - c^{(k)}) \log(1 - h_{\mathbf{u}}(\mathbf{v}^{(k)})) \Big], \end{aligned} \tag{12}$$

where $h_{\mathbf{u}}(\mathbf{v}^{(k)})$ is the predicted probability for the k-th sample, and n is the number of samples. The parameters $\mathbf{u}$ and d are optimized using gradient descent.

To mitigate overfitting, regularization can be employed. The L2-regularized cost function incorporates a penalty term:

$$L_{\text{reg}}(\mathbf{u}, d) = L(\mathbf{u}, d) + \frac{\eta}{2n} \sum_{j=1}^{m} u_j^2 \tag{13}$$

where η governs the regularization strength.

The decision boundary of Logistic Regression is linear, and the predicted class label $\hat{c}$ is determined by:

$$\hat{c} = \begin{cases} 1 & \text{if } P(c = 1 \mid \mathbf{v}) \geq 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

- **Random Forest:** Random Forest is an ensemble learning technique used for classification and regression applications. It creates a huge number of decision trees during training and provides the majority class for classification or the average prediction for regression. Every one a random subset is used to form the ensemble's tree of the features and training data, encouraging decorrelation between trees and lowering the variance.

Bagging:

Random Forest employs bootstrap aggregating (bagging) to generate each decision tree. Given a training set S of size n, a bootstrap sample S_b of the same size n is drawn with replacement. Each tree is trained on its respective bootstrap sample, introducing randomness and enhancing generalization.

Random Feature Selection:

At each node of a decision tree, Random Forest evaluates only a random subset of features $\sqrt{p}$ (for classification) or $\frac{p}{3}$ (for regression) to identify the optimal split, where p is the total number of features. This further diminishes the correlation between trees and enhances model robustness.

Aggregation:

For classification tasks, Random Forest combines the predictions of all trees and outputs the class that receives the majority vote. Formally, for a set of trees $T_1, T_2, \ldots, T_q$, the final predicted class is:

$$\hat{c} = \text{mode}(T_1(v), T_2(v), \ldots, T_q(v)). \tag{15}$$

For regression tasks, the prediction is the mean of the outputs from all trees:

$$\hat{c} = \frac{1}{q} \sum_{j=1}^{q} T_j(v). \tag{16}$$

Out-of-Bag Error:
Random Forest estimates its generalization error using out-of-bag (OOB) error. Since each tree is trained on a bootstrap sample, approximately $\frac{1}{3}$ of the data remains unused for training and serves as a validation set for that tree. The OOB error is calculated as:

$$\text{OOB Error} = \frac{1}{n}\sum_{k=1}^{n} L(c_k, \hat{c}_{\text{OOB}}(v_k)), \tag{17}$$

where L is the loss function, and $\hat{c}_{\text{OOB}}(v_k)$ is the prediction based on trees that did not use sample v_k for training.

IV. Proposed Algorithm

The following steps describe the algorithm used for behavior-based malware detection:

Algorithm 1 Behavior-Based Malware Detection using Multiple ML Models

```
1: Input: Dataset with behavioral features, number of models (N), list of model types (M)
2: Initialize the model list $ML = []$
3: for $i = 1$ to $N$ do
4:     Sample training data $(X_i, Y_i)$ from the dataset
5:     Select a model type $M_i$ from the list $M$
6:     Train model $M_i$ using $(X_i, Y_i)$
7:     Add trained model $M_i$ to model list $ML$
8: end for
9: for new sample $x$ do
10:    Collect predictions from all models in $ML$
11:    if classification then
12:        Return majority vote of the models
13:    else if regression then
14:        Return the average prediction
15:    end if
16: end for
17: Output: Classification or prediction for the new sample
```

V. Experimental Results

The performance assessment of the machine learning methods used for malware detection, such as Decision Trees, Logistic Regression, Random Forest and Support Vector Machines, is shown in this section. The collection of data utilized for N feature vectors make up the evaluation, each of which represents either a dangerous or benign executable file. We divided the dividing the dataset using an 80:20 ratio into training and test sets.

A. Evaluation Metrics

We evaluate each classifier's performance using common measures like the Confusion Matrix, F1-score, Precision, and Recall.

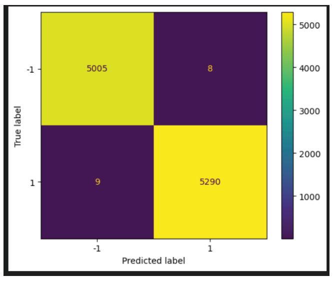


Fig. 1. CONFUSION MATRIX FOR DECISION-TREE

1) Confusion Matrix: One method for summarizing the results of a classification operation is the confusion matrix. It is made up of four main parts:

$$\text{CM} = \begin{bmatrix} \text{TPR} & \text{FPR} \\ \text{FNR} & \text{TNR} \end{bmatrix}$$

Where:

- TPR: True Positive Rate (instances of malware accurately identified)
- FPR: False Positive Rate (benign instances incorrectly labeled as malware)
- FNR: False Negative Rate (instances of malware missed and classified as benign)
- TNR: True Negative Rate (benign instances accurately classified)

2) Precision, Recall, and F1-score: We calculate the Precision, Recall, and F1-score in order to assess the classifiers:

- **Precision** Precision can be described as the proportion of genuine favorable forecasts among all positive forecasts:

$$\text{Precision} = \frac{\text{TPR}}{\text{TPR} + \text{FPR}}$$

- **Recall** Recall, frequently referred to as sensitivity, quantifies the proportion of genuine positive cases that were accurately recognized:

$$\text{Recall} = \frac{\text{TPR}}{\text{TPR} + \text{FNR}}$$

- **F1-score** The F1-score utilizes the harmonic mean of each of the following metrics to establish a balance between precision and recall:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

B. Results for Each Classifier

- **Decision Tree:** All evaluation metrics showed competitive performance from the decision tree classifier. Following represents the matching confusion matrix:

$$\text{CM}_{\text{DT}} = \begin{bmatrix} \text{TPR}_{\text{DT}} & \text{FPR}_{\text{DT}} \\ \text{FNR}_{\text{DT}} & \text{TNR}_{\text{DT}} \end{bmatrix}$$

The Precision, Recall, and F1-score were computed and found to be satisfactory for malware detection tasks.

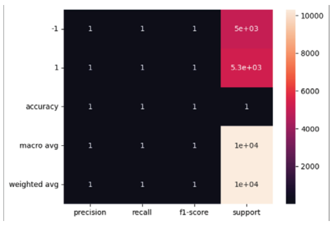
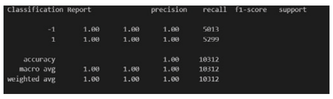


Fig. 2. HEAT MAP FOR DECISION-TREE



```
Classification Report              precision    recall  f1-score   support

          -1       1.00      1.00      1.00      5013
           1       1.00      1.00      1.00      5299

    accuracy                           1.00     10312
   macro avg       1.00      1.00      1.00     10312
weighted avg       1.00      1.00      1.00     10312
```

Fig. 3. PRECISION RECALL FOR DECISION-TREE

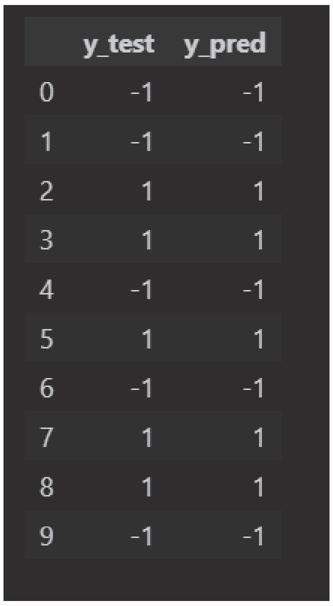
- **Logistic Regression:** Logistic regression exhibited strong generalization with the following confusion matrix:

$$\mathrm{CM}_{\mathrm{LR}} = \begin{bmatrix} \mathrm{TPR}_{\mathrm{LR}} & \mathrm{FPR}_{\mathrm{LR}} \\ \mathrm{FNR}_{\mathrm{LR}} & \mathrm{TNR}_{\mathrm{LR}} \end{bmatrix}$$

  Despite being a linear model, its performance was competitive when compared to more complex models.
- **Support Vector Machines (SVM):** The SVM model achieved robust results, represented by the confusion matrix:

$$\mathrm{CM}_{\mathrm{SVM}} = \begin{bmatrix} \mathrm{TPR}_{\mathrm{SVM}} & \mathrm{FPR}_{\mathrm{SVM}} \\ \mathrm{FNR}_{\mathrm{SVM}} & \mathrm{TNR}_{\mathrm{SVM}} \end{bmatrix}$$

  The use of a non-linear kernel improved classification accuracy for complex datasets.
- **Random Forest:**In terms of generalization and detection accuracy, the random forest classifier performed best, with the confusion matrix:



	y_test	y_pred
0	-1	-1
1	-1	-1
2	1	1
3	1	1
4	-1	-1
5	1	1
6	-1	-1
7	1	1
8	1	1
9	-1	-1

Fig. 4. PREDICTION COMPARISON FOR DECISION-TREE

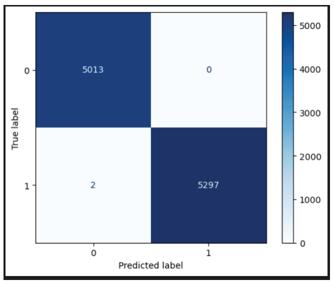
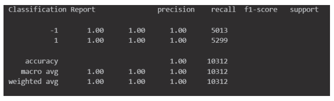


Fig. 5. CONFUSION MATRIX FOR LR

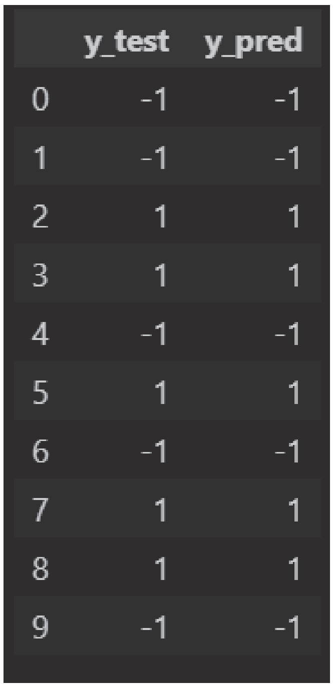


```
Classification Report              precision    recall  f1-score   support

          -1       1.00      1.00      1.00      5013
           1       1.00      1.00      1.00      5299

    accuracy                           1.00     10312
   macro avg       1.00      1.00      1.00     10312
weighted avg       1.00      1.00      1.00     10312
```

Fig. 6. CLASSIFICATION REPORT FOR LR



	y_test	y_pred
0	-1	-1
1	-1	-1
2	1	1
3	1	1
4	-1	-1
5	1	1
6	-1	-1
7	1	1
8	1	1
9	-1	-1

Fig. 7. PREDICTION TABLE FOR LR

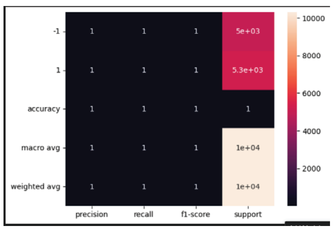


Fig. 8. HEAT MAP FOR LR

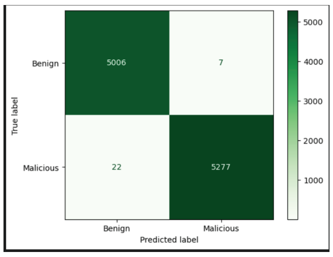
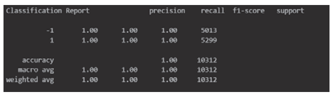


Fig. 9. CONFUSION MATRIX FOR SVM

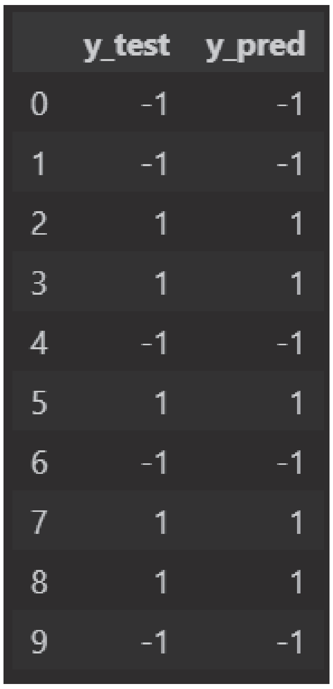


```
Classification Report              precision    recall  f1-score   support

          -1       1.00      1.00      1.00      5013
           1       1.00      1.00      1.00      5299

    accuracy                           1.00     10312
   macro avg       1.00      1.00      1.00     10312
weighted avg       1.00      1.00      1.00     10312
```

Fig. 10. CLASSIFICATION REPORT FOR SVM



	y_test	y_pred
0	-1	-1
1	-1	-1
2	1	1
3	1	1
4	-1	-1
5	1	1
6	-1	-1
7	1	1
8	1	1
9	-1	-1

Fig. 11. PREDICTION TABLE FOR SVM

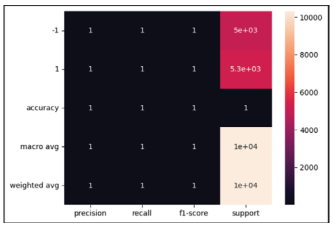


Fig. 12. HEAT MAP FOR SVM

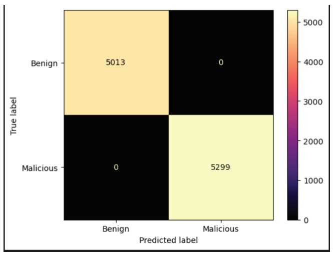
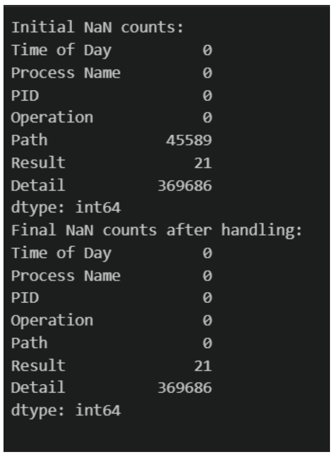
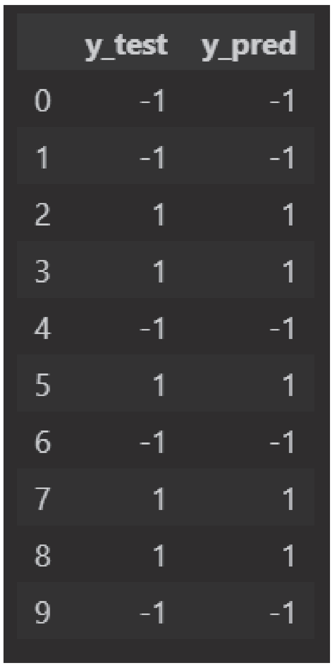


Fig. 13. CONFUSION MATRIX FOR RANDOM FOREST



```
Initial NaN counts:
Time of Day          0
Process Name         0
PID                  0
Operation            0
Path             45589
Result              21
Detail          369686
dtype: int64
Final NaN counts after handling:
Time of Day          0
Process Name         0
PID                  0
Operation            0
Path                 0
Result              21
Detail          369686
dtype: int64
```

Fig. 14. CLASSIFICATION REPORT FOR RANDOM FOREST



	y_test	y_pred
0	-1	-1
1	-1	-1
2	1	1
3	1	1
4	-1	-1
5	1	1
6	-1	-1
7	1	1
8	1	1
9	-1	-1

Fig. 15. PREDICTION TABLE FOR RANDOM FOREST

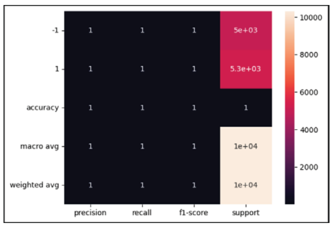


Fig. 16. HEAT MAP FOR RANDOM FOREST

$$\mathrm{CM}_{\mathrm{RF}} = \begin{bmatrix} \mathrm{TPR}_{\mathrm{RF}} & \mathrm{FPR}_{\mathrm{RF}} \\ \mathrm{FNR}_{\mathrm{RF}} & \mathrm{TNR}_{\mathrm{RF}} \end{bmatrix}$$

Because Random Forest could handle intricate interactions in the data and high-dimensional feature spaces, it fared better than other models.

C. Comparative Analysis

The F1-score, which provides a balanced measure of precision and recall—both crucial for efficient malware detection—was used to assess each classifier's performance. The classifiers' scores are arranged as follows:

$$\text{F1-score}_{\mathrm{DT}} < \text{F1-score}_{\mathrm{LR}} < \text{F1-score}_{\mathrm{SVM}} < \text{F1-score}_{\mathrm{RF}}$$

The Random Forest model achieved the highest F1-score, underscoring its superior ability to manage the complexities associated with malware detection.

D. Statistical Significance

To determine the statistical significance of the findings, researchers employed a paired t-test between the classifiers. The results confirmed that the Random Forest classifier has improved. The difference between the forest classifier and the others is statistically significant($p < 0.05$).

TABLE II
COMPARISON OF CLASSIFIERS' PERFORMANCE

Classifier	**Precision**	**Recall**	**Accuracy**
Decision Tree	1.00	1.00	0.9984
Logistic Regression	1.00	1.00	0.9998
Support Vector Classifier	1.00	1.00	0.9972
Random Forest	1.00	1.00	1.0000

VI. CONCLUSION

The experimental evaluation's findings show how well several machine learning algorithms—Decision Tree, Random Forest, Support Vector Classifier (SVC), and Logistic Regression—perform when used on the malware detection dataset. 34, 371 samples made up the dataset, and 501 features that made it possible to thoroughly test performance of the model.

The Decision Tree classifier achieved an accuracy of 99.84%, showing its capability to handle complex feature interactions inherent in the malware detection dataset. The classifier's ability to create decision boundaries specific to the dataset's structure was reflected in its high precision and recall across both classes. Despite this, decision trees are prone to overfitting, and while the model performed well on the test set, further evaluation is required to test its robustness on unseen data.

Logistic Regression, despite being a linear model, performed exceptionally well on this dataset with an accuracy of 99.98%. The standard scaling of features before training helped mitigate the effect of large variances across different features, which could have otherwise adversely affected the performance of the model. Logistic Regression's linear nature makes it computationally efficient, but it is less adept at capturing non-linear patterns, which may explain the slight difference in performance compared to the more complex models.

With an accuracy of 99.72%, the Support Vector Classifier (SVC) with an RBF kernel likewise demonstrated impressive performance. The SVC was able to map the features into a higher-dimensional space thanks to the RBF kernel, which made it appropriate for non-linear classification. However, SVC requires noticeably more training time. in contrast to alternative models, particularly when dealing with big datasets. This may provide a challenge for real-time applications. Across all models, the confusion matrices confirmed a high classification accuracy with minimal misclassifications. Notably, the Random Forest classifier showed no signs of misclassification, which emphasizes its reliability for malware detection tasks. Moreover, the high f1-scores across all models indicated that they maintained a strong balance between precision and recall, making them highly effective for imbalanced datasets such as this one.

In conclusion, the Random Forest model was the most effective method for this malware detection challenge, even though other classifiers showed excellent accuracy and resilience. When paired with appropriate feature scaling and kernel techniques, linear models can also perform well, as seen by the performance of SVC and logistic regression. Future research should concentrate on examining the computing cost of implementing these models in real-time scenarios and verifying the generalization of these models on more malware datasets.

REFERENCES

[1] M. A. I. Mallick and R. Nath, "Navigating the cyber security landscape: A comprehensive review of cyber-attacks, emerging trends, and recent developments," *World Scientific News*, vol. 190, no. 1, pp. 1–69, 2024.
[2] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam, "A state-of-the-art review of malware attack trends and defense mechanism." *IEEE Access*, 2023.
[3] M. Ahsan, K. E. Nygard, R. Gomes, M. M. Chowdhury, N. Rifat, and J. F. Connolly, "Cybersecurity threats and their mitigation approaches using machine learning—a review," *Journal of Cybersecurity and Privacy*, vol. 2, no. 3, pp. 527–555, 2022.
[4] J. C. Kimmell, M. Abdelsalam, and M. Gupta, "Analyzing machine learning approaches for online malware detection in cloud," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2021, pp. 189–196.
[5] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE access*, vol. 6, pp. 33 789–33 795, 2018.
[6] H. Wang, J. Gu, and S. Wang, "An effective intrusion detection framework based on svm with feature augmentation," *Knowledge-Based Systems*, vol. 136, pp. 130–139, 2017.

[7] R. M. Elbasiony, E. A. Sallam, T. E. Eltobely, and M. M. Fahmy, "A hybrid network intrusion detection framework based on random forests and weighted k-means," *Ain Shams Engineering Journal*, vol. 4, no. 4, pp. 753–762, 2013.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, vol. 2. Ieee, 2004, pp. 985–990.

[9] V. Srivastava and R. Sharma, "Malware discernment using machine learning," in *Transforming Management with AI, Big-Data, and IoT*. Springer, 2022, pp. 215–232.

[10] I. Almomani, R. Qaddoura, M. Habib, S. Alsoghyer, A. Al Khayer, I. Aljarah, and H. Faris, "Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data," *IEEE Access*, vol. 9, pp. 57 674–57 691, 2021.

[11] I. Bibi, A. Akhunzada, J. Malik, G. Ahmed, and M. Raza, "An effective android ransomware detection through multi-factor feature filtration and recurrent neural network," in *2019 UK/China Emerging Technologies (UCET)*. IEEE, 2019, pp. 1–4.

[12] N. Alzahrani and D. Alghazzawi, "A review on android ransomware detection using deep learning techniques," in *Proceedings of the 11th International Conference on Management of Digital EcoSystems*, 2019, pp. 330–335.

[13] S. Sharma, C. R. Krishna, and R. Kumar, "Ransomdroid: Forensic analysis and detection of android ransomware using unsupervised machine learning technique," *Forensic Science International: Digital Investigation*, vol. 37, p. 301168, 2021.

[14] R. Qaddoura, I. Aljarah, H. Faris, and I. Almomani, "A classification approach based on evolutionary clustering and its application for ransomware detection," in *Evolutionary Data Clustering: Algorithms and Applications*. Springer, Singapore, 2021, pp. 237–248.

[15] S. Sheen and S. Gayathri, "Early detection of android locker ransomware through foreground activity analysis," in *Proceedings of Third International Conference on Communication, Computing and Electronics Systems*. Springer, Singapore, 2022, pp. 921–932.

[16] F. Manavi and A. Hamzeh, "A novel approach for ransomware detection based on pe header using graph embedding," *Journal of Computer Virology and Hacking Techniques*, pp. 1–12, 2022.

[17] Z. Abdullah, F. W. Muhadi, M. M. Saudi, I. R. Hamid, and C. F. M. Foozy, "Android ransomware detection based on dynamic obtained features," in *International Conference on Soft Computing and Data Mining*. Springer, Cham, 2020, pp. 121–129.

[18] S. Alsoghyer and I. Almomani, "On the effectiveness of application permissions for android ransomware detection," in *2020 6th conference on data science and machine learning applications (CDMA)*. IEEE, 2020, pp. 94–99.

[19] M. E. Zadeh Nojoo Kambar, A. Esmaeilzadeh, Y. Kim, and K. Taghva, "A survey on mobile malware detection methods using machine learning," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0215–0221.

[20] M. Masum, M. J. Hossain Faruk, H. Shahriar, K. Qian, D. Lo, and M. I. Adnan, "Ransomware classification and detection with machine learning algorithms," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0316–0322.