INTERNATIONAL JOURNAL OF RESEARCH, EXPLORATION & KNOWLEDGE (IJREK)
Official Scholarly Journal Portal: **https://www.ijrek.org** | DOI: **https://doi.org/10.5281/zenodo.23641151**

MALWARE DETECTION USING MACHINE LEARNING

Mrs.Dalavi.M.T[1], Snehal Ghongade[2]

[1] Faculty Guide & Supervisor | [2] Dattakala Group Of Institutions Faculty Of Engineering Swami - Chincholi tal. Daund dist. Pune, India

Corresponding author: **snehalghongade3@gmail.com**

ARTICLE ID	ARTICLE TYPE	RESEARCH AREA	PUBLICATION
IJREK-2026-00155	Research Article	Machine Learning & Deep Learning	Volume 1, Issue Issue 1 (2026)
RECEIVED	**REVISED**	**ACCEPTED**	**PUBLISHED**
23 October 2026	30 October 2026	26 October 2026	02 November 2026

Abstract

Malware is a major cybersecurity threat because malicious files and programs can steal information, damage systems, disrupt services, and provide unauthorized access. Traditional signature-based detection methods are useful for known threats but may struggle to identify previously unseen or modified malware. This paper presents a machine-learning-based approach for classifying files or executable samples as benign or malicious using extracted features. The proposed workflow includes dataset collection, data cleaning, feature preparation, model training, testing, and performance evaluation. Candidate classifiers may include Decision Tree, Random Forest, Support Vector Machine, or other suitable models; the final model should be selected using measured results on a held-out test set. Evaluation should report metrics such as precision, recall, F1-score, accuracy, and confusion matrix. The approach is intended to support automated malware screening, while recognizing that model performance depends on dataset quality, feature selection, and the ability of malware to change over time. All experimental values in this draft are placeholders and must be replaced with results from the student's actual implementation.

Keywords: *Malware detection; machine learning; cybersecurity; classification; feature extraction; Random Forest; malware analysis*

I. INTRODUCTION

A. Background / Context Malware, or malicious software, includes viruses, worms, Trojans, ransomware, spyware, and other programs designed to perform harmful or unauthorized actions. Malware can spread through email attachments, websites, downloaded applications, removable devices, and compromised software. As cyber threats continue to evolve, systems need effective ways to identify suspicious files before they cause harm. Machine learning (ML) can learn patterns from labelled examples and use those patterns to classify new samples. B. Problem Statement Signature-based antivirus tools primarily identify threats using known patterns. This can be less effective when a malicious program is new, packed, obfuscated, or modified to avoid an existing signature. Manual analysis can also require specialist time and expertise. Therefore, a data-driven detection approach is needed to examine relevant file or program features and estimate whether a sample is benign or malicious. Such a model must be evaluated carefully to avoid excessive false alarms or missed threats. C. Objective / Aim To study how machine-learning techniques can be applied to malware detection. To prepare a labelled dataset and extract or select useful features for classification. To train and compare suitable classification algorithms using a reproducible workflow. To evaluate the selected model using accuracy, precision, recall, F1-score, and a confusion matrix. To provide a simple output that identifies a sample as benign or potentially malicious, with appropriate limitations. D. Significance / Contribution The proposed work provides a structured workflow for using supervised learning in malware classification. Its contribution should be demonstrated through the student's actual dataset, selected features, model comparison, and measured evaluation results. The system is intended as a decision-support tool and should not be treated as a replacement for layered security controls or expert malware analysis. E. Paper Organization The remainder of this paper reviews related work, describes the proposed system and methodology, outlines implementation details, presents the evaluation plan and results, discusses limitations and future scope, and concludes the study.

II. LITERATURE REVIEW / RELATED WORK

A. Overview of Existing Research Malware detection research commonly explores signature-based analysis, static analysis, dynamic analysis, and machine-learning-based classification. Static analysis examines characteristics available without running a sample, such as file metadata, imported functions, byte patterns, or executable structure. Dynamic analysis observes behavior while a sample executes in a controlled environment, such as file changes, network activity, or system calls. ML methods can use features from either approach to learn distinctions between benign and malicious samples. B. Machine-Learning Techniques Common supervised classifiers include Decision Trees, Random Forests, Support Vector Machines, Logistic Regression, and gradient-boosting methods. The suitability of each model depends on the dataset size, feature types, class balance, computational requirements, and evaluation protocol. Neural networks may also be considered when the dataset and feature representation justify their complexity. No algorithm should be described as best unless the experiment supports that conclusion. C. Identified Research Gap A model that performs well on one dataset may not generalize to malware families, file formats, or time periods not represented in training data. Class imbalance, duplicate samples, data leakage, changing attacker behavior, and obfuscation can distort evaluation. A useful student project should document its data source, feature choices, split strategy, metrics, and limitations so that reported performance can be interpreted correctly. D. Contribution of the Proposed Work This work proposes a reproducible pipeline that preprocesses labelled samples, trains one or more classifiers, compares their performance on held-out data, and reports the results transparently. The exact dataset, feature set, model, and contribution must be aligned with the implementation completed by the student.

III. PROPOSED SYSTEM AND METHODOLOGY

TABLE III

Input	Processing	Output
Labelled dataset	Validation and cleaning	Prepared data
Prepared training data	Feature processing and model fitting	Trained classifier
Held-out test data	Prediction and metric calculation	Evaluation results
New supported sample/features	Same preprocessing + classifier	Benign / potentially malicious prediction

Fig. 1. Proposed malware detection workflow. B. System Design A modular design separates data handling, feature processing, model training, evaluation, and the user interface. A Python implementation may use pandas and NumPy for data handling and scikit-learn for preprocessing and classification. A web interface is optional and should only be described if implemented. Training and test data must be separated before fitting preprocessing steps to prevent information from the test set leaking into training. C. Main System Modules D. System Algorithm Step 1: Load a documented, labelled dataset from a legitimate research or educational source. Step 2: Validate labels, inspect class distribution, remove or investigate duplicates, and handle missing data. Step 3: Split the dataset into training and test sets using a documented strategy. Step 4: Fit required preprocessing and feature-selection steps using training data only. Step 5: Train selected classification algorithms on the training data. Step 6: Predict labels for the held-out test set. Step 7: Calculate accuracy, precision, recall, F1-score, and the confusion matrix. Step 8: Compare results, select a model based on measured performance and project requirements, and document limitations. E. Feature Extraction Features depend on the selected dataset and detection approach. Static executable features may include file size, header fields, imported libraries, section information, entropy, or byte-derived statistics. Dynamic features may include observed API calls, process activity, file-system changes, or network behavior collected in a controlled analysis environment. Only features actually present in the chosen dataset should be claimed in the final paper. F. Working Principle / System Flow The dataset is first prepared and labelled. The model learns patterns from the training examples and is then tested on samples not used during training. For a new sample, the system applies the same feature extraction and preprocessing pipeline, passes the resulting feature vector to the trained classifier, and displays the predicted class. Suspicious files should be handled safely and predictions should be treated as indicators rather than definitive proof. G. Advantages of the Proposed System Can learn patterns from labelled samples rather than relying only on manually written signatures. Can compare multiple classifiers using a consistent evaluation process. Can automate initial screening of samples represented by the training data. Can provide quantitative evaluation through precision, recall, F1-score, and confusion matrix. Can be extended with new datasets and features after careful retraining and validation.

TABLE I

Module	Description
Dataset Module	Loads labelled benign and malicious samples or a prepared feature dataset.
Preprocessing Module	Checks missing values, duplicates, data types, labels, and class distribution.
Feature Module	Selects or extracts relevant static or behavioral features supported by the dataset.
Training Module	Trains selected ML classifiers using training data and a documented configuration.
Evaluation Module	Calculates metrics and confusion matrix using held-out test data.
Prediction / Interface Module	Displays a predicted class or risk score with a clear warning about model limitations.

TABLE II

Component	Proposed specification (confirm against actual work)
Programming Language	Python
Data Processing	pandas, NumPy
Machine Learning	scikit-learn or another library actually used
Dataset	[Insert dataset name, source, version, and access date]
Feature Set	[Insert features actually used]
Interface	[Command-line, notebook, desktop, or web interface actually implemented]
Environment	[Operating system, Python version, hardware, and package versions]

TABLE IV

Test ID	Test condition	Expected behaviour
T1	Known benign sample represented by valid features	Model returns a benign prediction when supported by learned patterns.
T2	Known malicious sample represented by valid features	Model returns a malicious prediction when supported by learned patterns.
T3	Missing or malformed feature input	System reports invalid input rather than crashing.
T4	Unseen or out-of-distribution sample	System displays the prediction with an uncertainty/limitations warning.
T5	Dataset or model loading failure	System reports a safe error and does not produce a misleading result.

TABLE V

Metric	Measured value
Accuracy	[Enter measured value]
Precision (malware class)	[Enter measured value]
Recall (malware class)	[Enter measured value]
F1-score (malware class)	[Enter measured value]
False-positive rate	[Enter measured value]
Average prediction time	[Enter measured value and test conditions]

TABLE VI

Approach	Potential strength	Potential limitation
Signature-based detection	Effective for recognized signatures	May miss novel or modified variants
Rule-based detection	Transparent and straightforward for known patterns	Rules need maintenance and may not generalize
ML-based classification	Learns patterns from labelled features	Depends on data quality and may fail on unseen threats

IV. IMPLEMENTATION DETAILS

A. Technology and Component Specification B. System Implementation The implementation should load the selected dataset, verify target labels, and prepare the features for training. Categorical fields may require encoding, while numeric fields may require scaling depending on the selected algorithm. Any resampling or class-weighting strategy should be applied only to the training data. The trained pipeline and model configuration should be saved with version details so that predictions can be reproduced. C. Dataset Description Document the dataset's name, source, license or usage terms, number of benign and malicious samples, feature count, label definitions, collection period if known, and any exclusions. Avoid publishing sensitive samples or running malware outside an isolated and authorized environment. If a public benchmark is used, cite its original paper or official source. D. Data Flow

V. RESULTS AND ANALYSIS

This section must contain results from the student's actual experiments. Do not use invented performance numbers. Report the split strategy, class distribution, selected model settings, and test-set metrics. If the model has not yet been implemented, keep the following tables as templates and complete them after testing. A. Test Cases

B. Performance Evaluation — Fill with Actual Results C. Discussion Discuss the measured results, including the balance between false positives and false negatives. A false positive may incorrectly flag a benign file, while a false negative may fail to identify malicious software. Accuracy alone may be misleading when classes are imbalanced, so precision, recall, F1-score, and the confusion matrix should also be considered. Compare models only when they use the same data split and evaluation procedure. D. Comparative Analysis

VI. LIMITATIONS AND FUTURE SCOPE

A. Limitations Model performance depends on the quality, diversity, and freshness of the dataset. A model may learn dataset-specific artifacts rather than general malware characteristics. Packing, obfuscation, and new malware families may reduce detection performance. False positives and false negatives cannot be eliminated entirely. Dynamic analysis requires a controlled, isolated environment and appropriate authorization. Results from a benchmark dataset may not represent real-world deployment performance. B. Future Scope Evaluate on newer and temporally separated datasets to assess generalization. Add explainability methods to help analysts understand influential features. Explore ensemble learning or deep learning only where justified by data and evaluation. Add drift monitoring and a documented process for updating the model. Integrate the detector into a safe file-scanning workflow with human review. Test robustness against data leakage, class imbalance, and previously unseen malware families.

VII. CONCLUSION

This paper outlines a machine-learning-based workflow for malware detection using labelled data, feature preparation, supervised classification, and quantitative evaluation. The approach can assist in identifying suspicious samples represented by the training data, but its reliability depends on dataset quality, feature selection, evaluation design, and changes in attacker techniques. A complete student paper must include the actual dataset details, implementation decisions, measured results, and verified references. The system should be used as one layer of a broader security process rather than as a guarantee that a file is safe.

REFERENCES

Add and verify the exact sources you used. The following are suggested foundational references; check the publication details and use the citation style required by your college.

[1] R. S. Saxe and H. S. Berlin, “Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features,” 2015.

[2] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, “Data Mining Methods for Detection of New Malicious Executables,” Proceedings of the IEEE Symposium on Security and Privacy, 2001.

[3] scikit-learn developers, “scikit-learn: Machine Learning in Python,” official documentation. [Add access date and URL.]

[4] [Add the original paper or official source for the dataset used in this project.]

[5] [Add recent, relevant malware-detection research papers that you have read and verified.] STUDENT CHECKLIST BEFORE SUBMISSION Replace all bracketed placeholders with actual student, guide, department, dataset, and implementation details. Do not claim a model, feature, interface, or experiment unless it was actually implemented. Replace all result placeholders with measured values from your own testing. Insert your own architecture diagram, screenshots, confusion matrix, and result charts where appropriate. Verify every reference and cite any borrowed idea, figure, dataset, or text. Check formatting, plagiarism/similarity, and guide approval before submission.

Publication archival note: Official scholarly contribution published under CC BY 4.0 International License. Permanent record on **https://www.ijrek.org** | Article ID: IJREK-2026-00155