Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.052835

ARTICLE

Fine-Tuning Cyber Security Defenses: Evaluating Supervised Machine Learning Classifiers for Windows Malware Detection

Islam Zada[1,*], Mohammed Naif Alatawi[2], Syed Muhammad Saqlain[1], Abdullah Alshahrani[3], Adel Alshamran[4], Kanwal Imran[5] and Hessa Alfraihi[6]

[1]Department of Software Engineering, International Islamic University, Islamabad, 25000, Pakistan

[2]Information Technology Department, Faculty of Computers and Information Technology, University of Tabuk, Tabuk, 71491, Saudi Arabia

[3]Department of Computer Science and Artificial Intelligence, College of Computer Science and Engineering, University of Jeddah, Jeddah, 21493, Saudi Arabia

[4]Department of Cybersecurity, College of Computer Science and Engineering, University of Jeddah, Jeddah, 21493, Saudi Arabia

[5]Department of Computer Science, University of Peshawar, Peshawar, 25121, Pakistan

[6]Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, 11671, Saudi Arabia

*Corresponding Author: Islam Zada. Email: islam.zada@iiu.edu.pk

Received: 16 April 2024 Accepted: 09 July 2024 Published: 15 August 2024

ABSTRACT

Malware attacks on Windows machines pose significant cybersecurity threats, necessitating effective detection and prevention mechanisms. Supervised machine learning classifiers have emerged as promising tools for malware detection. However, there remains a need for comprehensive studies that compare the performance of different classifiers specifically for Windows malware detection. Addressing this gap can provide valuable insights for enhancing cybersecurity strategies. While numerous studies have explored malware detection using machine learning techniques, there is a lack of systematic comparison of supervised classifiers for Windows malware detection. Understanding the relative effectiveness of these classifiers can inform the selection of optimal detection methods and improve overall security measures. This study aims to bridge the research gap by conducting a comparative analysis of supervised machine learning classifiers for detecting malware on Windows systems. The objectives include Investigating the performance of various classifiers, such as Gaussian Naïve Bayes, K Nearest Neighbors (KNN), Stochastic Gradient Descent Classifier (SGDC), and Decision Tree, in detecting Windows malware. Evaluating the accuracy, efficiency, and suitability of each classifier for real-world malware detection scenarios. Identifying the strengths and limitations of different classifiers to provide insights for cybersecurity practitioners and researchers. Offering recommendations for selecting the most effective classifier for Windows malware detection based on empirical evidence. The study employs a structured methodology consisting of several phases: exploratory data analysis, data preprocessing, model training, and evaluation. Exploratory data analysis involves understanding the dataset's characteristics and identifying preprocessing requirements. Data preprocessing includes cleaning, feature encoding, dimensionality reduction, and optimization to prepare the data for training. Model training utilizes various supervised classifiers, and their performance is evaluated using metrics such as accuracy, precision, recall, and F1 score. The study's outcomes comprise a comparative analysis of supervised machine learning classifiers for Windows malware detection. Results reveal the effectiveness and efficiency of each classifier in detecting different types of malware. Additionally, insights into their strengths and limitations provide practical guidance for enhancing cybersecurity defenses. Overall, this research contributes to advancing malware detection techniques and bolstering the security posture of Windows systems against evolving cyber threats.

KEYWORDS

Security and privacy challenges in the context of requirements engineering; supervised machine learning; malware detection; windows systems; comparative analysis; Gaussian Naïve Bayes; K Nearest Neighbors; Stochastic Gradient Descent Classifier; Decision Tree

1 Introduction

The escalating threat of malware in contemporary digital ecosystems, especially within Windows operating environments, underscores the urgent need for robust detection mechanisms. Malicious software, spanning a spectrum from viruses to ransomware, poses severe risks including data breaches, system compromise, and operational disruptions. Consequently, the development of effective malware detection methodologies has become paramount for safeguarding systems and data integrity.

Supervised machine learning offers a promising avenue for malware detection, leveraging labeled datasets to train classifiers capable of discerning malicious patterns and behaviors. Among the diverse array of supervised learning algorithms, Gaussian Naïve Bayes, K Nearest Neighbors (KNN), Stochastic Gradient Descent Classifier (SGDC), and Decision Tree have emerged as notable contenders for malware detection. However, a comprehensive comparative analysis of these classifiers, specifically tailored for Windows malware detection, is notably absent from existing literature.

The malware industry, like any other software industry, is a stable, well-organized, and well-funded market and is taking measures to evade traditional security measures. To solve the issue of malware attacks on Windows machines, Microsoft decided to take countermeasures to detect possible attacks before they happened and then make their system more secure and more durable [1]. This is an essential measure to take because once the malware successfully hits the system and manages to take control of the system, the valuable information of the end user or a business or sensitive information may be at stake which may result in a drastic drop in the clients' trust on the Microsoft's system. So, Microsoft challenged the data scientists and data analysts from across the globe to make the prediction on the data provided by them which is the real data by hiding the end user's private details. There are numerous data-driven techniques imposed as research work to determine the in-time possibility of malware attacks on machines to better tackle it to minimize the loss associated with the attack [2]. Some of these techniques work on executable processes while others draw out the patterns from the malware data to match the programs to check whether it is malware. Our proposed work takes the data which is System configurations such as 'Machine Version', 'Operating System (OS) version', 'Processor type', 'firewall', etc., and predicts the malware attack using supervised Machine Learning techniques. We applied various classification techniques to the given data and compared the outcomes of these techniques during the analysis. If we talk about the motive for this research, Microsoft, who offered a \$25,000 reward on the international data science competition website "Kaggle," is the motivating factor behind it. The threat posed by malware to Windows operating systems is the highest of all the companies competing in this industry, Microsoft likewise handled this issue seriously. All the machines that are affected by the malware threats, around 87% of them are Windows machines, which is a huge figure and a serious indication for Microsoft to consider security improvements in their operating systems. The task here is to analyze the system configuration and build the model which in turn would be able to predict the probability of malware attack on the system with provided facts and system configurations [3–5]. This prediction is for the Windows operating system as the data gathered is provided by Microsoft from their Windows operating system users. The dataset is provided by Microsoft from their Windows machine users by considering the end user privacy, so the identification is hidden. The data is sampled by taking a major proportion of the machines hit by malware. Also, during the exploration of the data, we figured out that the data is balanced and contains almost equal amounts of both target classes, the one in which malware was detected and the one in which malware wasn't detected. This makes sure that our classification is unbiased.

The main contributions of this article are as follows:

1. Predicting whether a Windows Personal Computer (PC) would probably decline prey to a malware assault.
2. Prediction of malware attack using system configuration data, such as machine version, OS version, processor type, and a firewall based on machine learning.
3. On the feature set, feature selection techniques have been used to extract the most significant features, which can reduce computation requirements without affecting the detection performance of machine learning algorithms.
4. Comparing the accuracy of the predicted results from various machine learning classification algorithms, including K Nearest Neighbors (KNN), Support Vector Machine (SVM), and others.

1.1 The Necessity of Performing a Comparative Analysis

Performing a comparative analysis of supervised machine learning-based Windows malware detection methods is essential for several reasons.

Firstly, the landscape of cybersecurity threats, particularly those targeting Windows systems, is constantly evolving. As attackers develop more sophisticated malware variants, it becomes crucial for security researchers and practitioners to assess the effectiveness of different detection approaches. A comparative analysis allows for the systematic evaluation of multiple supervised machine learning classifiers, providing insights into their performance in detecting a diverse range of malware samples.

Secondly, by conducting a comparative analysis, researchers can identify the strengths and weaknesses of each classifier in the context of Windows malware detection. Different classifiers may excel in certain scenarios based on factors such as dataset characteristics, feature extraction methods, and model complexity. Through rigorous evaluation using standardized metrics, such as accuracy, True Positive Rate (TPR), and False Positive Rate (FPR), researchers can determine which classifiers offer the highest levels of detection accuracy and robustness across various malware families and attack vectors.

Lastly, a comparative analysis facilitates the selection of the most effective supervised machine learning approach for Windows malware detection in practical settings. By identifying the top-performing classifiers, cybersecurity professionals can make informed decisions when deploying detection systems in real-world environments. This ensures that resources are allocated efficiently and that organizations can effectively defend against evolving threats. Moreover, the insights gained from the comparative analysis contribute to the advancement of malware detection techniques, driving innovation in the field of cybersecurity research and enabling the development of more resilient defense mechanisms.

1.2 Contribution to the Windows Malware Detection Domain

In addition to conducting comprehensive experiments, this study makes a significant contribution to the Windows malware detection domain by advancing our understanding of the effectiveness of supervised machine learning techniques in combating malware threats targeting Windows systems. By evaluating and comparing multiple supervised learning classifiers, including Support Vector Machine (SVM), K Nearest Neighbors (KNN), Gaussian Naïve Bayes, and Decision Tree, this research provides valuable insights into the strengths and limitations of different detection approaches. Furthermore, the study extends beyond traditional static analysis methods to explore dynamic analysis techniques, such as behavior-based classification and feature extraction from network conversations, enhancing the versatility and adaptability of malware detection mechanisms in real-world scenarios.

Moreover, this research contributes to the development of robust and reliable malware detection frameworks tailored specifically for Windows environments. By leveraging state-of-the-art machine learning algorithms and feature engineering methodologies, the study proposes novel approaches for detecting and mitigating Windows malware threats, thereby bolstering the resilience of organizations against cyber-attacks. Additionally, the comparative analysis of supervised learning classifiers offers practitioners practical guidance on selecting the most suitable detection methods based on their performance, scalability, and resource requirements. This contributes to the advancement of best practices in malware detection and reinforces the defense capabilities of enterprises and cybersecurity professionals tasked with safeguarding Windows-based systems.

Furthermore, the findings of this study serve as a foundation for future research endeavors aimed at addressing emerging challenges and evolving threats in the Windows malware detection domain. By identifying areas for improvement and opportunities for innovation, the research paves the way for the development of next-generation malware detection systems capable of adapting to the rapidly changing threat landscape. Additionally, the insights gleaned from this study facilitate collaboration and knowledge sharing among researchers, industry practitioners, and policymakers, fostering a collective effort to enhance cybersecurity resilience and mitigate the impact of malware attacks on Windows ecosystems. Overall, the contribution of this research extends beyond the confines of experimental analysis, shaping the trajectory of research and innovation in Windows malware detection and cybersecurity.

The purpose of this work is to investigate in-depth the effectiveness of supervised machine learning classifiers for Windows malware detection. After the current "introduction section", Section 2 summarizes the body of research on machine learning-based malware detection and offers an overview of related work. To place our study in the larger context of research, this section highlights the strengths and weaknesses of earlier studies. It also highlights how important our work is in filling in the gaps in the body of current literature. The steps of exploratory data analysis, data preprocessing, model training, and evaluation are described in Section 3 of our approach. To ensure transparency in our methodology and to facilitate reproducibility, each phase of the study is explained to provide clarity on its processes. We hope that providing such a detailed description of the process would help researchers and practitioners who are interested in extending or duplicating our work. Section 4 then goes over the findings and discussion, along with a performance comparison of the classifiers. Here, we explore our study's empirical results, providing insight into the efficacy and efficiency of each classifier in identifying various malware kinds. Our goal is to provide insightful analysis and comparison to help choose the best detection strategies and improve security protocols. Section 5 concludes with a summary of the major discoveries and suggestions for future lines of inquiry. We hope to support ongoing efforts to strengthen cybersecurity defense against changing threats by summarizing the study's findings and considering their consequences. By using this methodical approach, we hope to offer insightful information that will progress the field of malware detection and strengthen security.

2 Related Work

An extensive review of the related work in machine learning-based malware detection is given in this section. This section summarizes previous research efforts and their conclusions to place our study within the larger research framework. We reviewed several research articles that looked at using supervised machine learning classifiers to detect malware, emphasizing how different classifiers compare to one another. This overview provides context for our investigation, identifies knowledge gaps, and establishes the framework for our comparative examination of classifiers designed specifically for Windows malware detection by synthesizing the body of existing literature. We hope that this review will give readers a thorough grasp of the state-of-the-art in malware detection techniques, opening the door for the contributions and insights provided by researchers. A detailed overview is provided in the subsequent sections followed by Table 1 which summarizes the related work overview in a comprehensive and scholarly manner.

Table 1: Related work overview

	Related work 1	Related work 2	Related work 3	My project
Category	Classification	Classification	Classification	Classification/ prediction
Comparative analysis done	✓	✓	✓	✓
Technique used	Malicious websites prediction	recurrent neural networks	Self-organizing feature maps	Your choice (SVM & KNN)
Type	Competition	Research article	Research article	Competition
Description	Classification of malware into families	Classification of malware into families	Classification of malware into families	Malware attack prediction
Worked with	Malware dataset	Executable software	Executable software	System configuration
Training type	Supervised	Supervised	Unsupervised	Supervised

2.1 Microsoft Malware Classification Challenge (BIG 2015)

This was a challenge for the data science community hosted by Microsoft in 2015. The problem was the vast amount of data files that needed to be evaluated for potential malware threats to evade detection, malware authors introduce polymorphism to the malicious components [6]. This means that malicious files belonging to the same malware "family", with the same forms of malicious behavior, are constantly modified and/or obfuscated using various tactics, such that they look like many different files. For this challenge, Microsoft provided the malware dataset and required it to be classified into families [7].

2.2 Early-Stage Malware Prediction Using Recurrent Neural Networks

In [8], the authors analyzed that static malware analysis is well-suited to endpoint anti-virus systems as it can be conducted quickly by examining the features of an executable piece of code and matching it to previously observed malicious code. This is the first time general types of a malicious file have been predicted to be malicious during execution rather than using a complete activity log file post-execution and enables cyber security endpoint protection to be advanced to use behavioral data for blocking malicious payloads rather than detecting them post-execution and having to repair the damage [9]. However, static code analysis can be vulnerable to code obfuscation techniques. Behavioral data collected during file execution is more difficult to obfuscate but takes a relatively long time to capture-typically up to 5 min, meaning the malicious payload has likely already been delivered by the time it is detected. In [10,11], the authors investigated the possibility of predicting whether an executable is malicious based on a short snapshot of behavioral data. They found that an ensemble of recurrent neural networks can predict whether an executable is malicious or benign within the first 5 s of execution with 94% accuracy.

2.3 Malware Classification Using Self-Organizing Feature Maps and Machine Activity Data

This article is about the use of machine activity metrics to automatically distinguish between malicious and trusted portable executable software samples. The motivation stems from the growth of cyber-attacks using techniques that have been employed to surreptitiously deploy Advanced Persistent Threats (APTs). APTs are becoming more sophisticated and able to obfuscate much of their identifiable features through encryption, custom code bases, and in-memory execution [12–14]. Machine learning offers a way to potentially construct malware classifiers to detect new and variant malware to address this issue [15–17]. Numerous machine learning-based methods have been put forth in the literature using supervised and unsupervised algorithms [18,19]. Two key conclusions are drawn after analyzing the suggested machine learning-based detection methods [20,21].

The hypothesis is that we can produce a high degree of accuracy in distinguishing malicious from trusted samples using machine learning with features derived from the inescapable footprint left behind on a computer system during execution. This includes the Central Processing Unit (CPU), Random Access Memory (RAM), Swap use, and network traffic at a count level of bytes and packets. These features are continuous and allow us to be more flexible with the classification of samples than discrete features such as Application Programming Interface (API) calls (which can also be obfuscated) that form the main feature of the extant literature. We use these continuous data and develop a novel classification method using Self Organizing Feature Maps to reduce overfitting during training through the ability to create unsupervised clusters of similar "behavior" that are subsequently used as features for classification, rather than using raw data.

Comparison of accuracy achieved by various supervised machine learning techniques for Windows malware detection. Our study demonstrates superior performance with an accuracy of 99.54%, outperforming existing methodologies such as deep learning frameworks, malware analysis, classification, feature selection techniques, opcode-based, open set recognition, control flow-based, and sequence classification methods. The comparative analysis presented in Table 2 highlights the superior performance of our study in the domain of Windows malware detection compared to existing literature. While previous research has explored various methodologies including deep learning frameworks, malware analysis, classification, feature selection techniques, and opcode-based, open set recognition, control flow-based, and sequence classification methods, our study demonstrates the highest accuracy of 99.54%. This indicates the effectiveness of our chosen supervised machine learning techniques in accurately identifying and classifying Windows malware, thereby contributing significantly to enhancing cybersecurity measures for Windows systems. Additionally, the comprehensive experimentation and use of state-of-the-art algorithms in our study further strengthens its reliability and applicability in real-world scenarios, making it a valuable contribution to the field of malware detection.

Table 2: Comparative analysis of supervised machine learning techniques

Study	Method	Accuracy (%)
The current study (2024)	Supervised learning	99.54
Hardy et al. [22] (2016)	Deep learning framework	96.3
Gandotra et al. [23] (2016)	Malware analysis and classification	56–64
Yuxin et al. [24] (2020)	CNN (convolutional neural network)	95.0 (average)
Srinivasan et al. [25] (2023)	Ensemble classification-based machine learning	97.8
Tayyab et al. [26] (2022)	Deep learning-based classification	98.2
Oak et al. [27] (2019)	Deep learning methods	93.2
Cakir et al. [28] (2018)	Deep learning	92.5
Verma et al. [29] (2024)	Ensemble machine learning approach	94.7
Usman et al. [30] (2021)	API calls extraction	91.6
Zhang [31] (2019)	Feature selection with principal component analysis (PCA)	90.3
Apruzzese et al. [32] (2018)	Deep learning	88.9
Kumar et al. [33] (2021)	Fine-tune convolution neural network, transfer learning	85.7
Seneviratne et al. [34] (2022)	Self-supervised vision transformers	87.2
Zhao et al. [35] (2014)	Control-flow construct feature of software (Knns)	89.4
Lu et al. [36] (2019)	Sequence and statistics features combined architecture for malware detection	86.39

3 Methodology

The methodology used in this study is described in Section 3, which offers an organized way to look into how well-supervised machine learning classifiers work for detecting Windows malware. Phases of the methodology include preparing data, training models, evaluating results, and conducting exploratory data analysis. Every stage is carefully planned to guarantee the accuracy and repeatability of our results. Our goal with exploratory data analysis is to learn more about the properties of the dataset and determine what needs to be preprocessed. Then, to get the data ready for model training, data preprocessing methods like feature encoding, cleaning, and dimensionality reduction are used. The choice and training of several supervised classifiers, such as Gaussian Naïve Bayes, K Nearest Neighbors, Stochastic Gradient Descent Classifier (SGDC), and Decision Tree, are then covered in detail. Because the same approach was also used by [37–39] for similar problems. Lastly, a variety of metrics are used to evaluate the model's performance to determine its accuracy, precision, recall, and F1 score. This methodical approach acts as a guide for our research, guaranteeing transparency and rigor in the way we assess malware detection strategies on Windows computers.

We address the critical aspect of the experimental setup and dataset characteristics to ensure transparency and reproducibility of our study. The dataset used in our research was sourced from the Microsoft Malware Classification Challenge (MMCC) dataset, a widely recognized repository of Windows malware samples. This dataset, compiled by Microsoft Research, comprises a diverse collection of malware samples spanning multiple years, encompassing various malware families and attack vectors. The MMCC dataset is publicly available and has been extensively used in academic research for evaluating malware detection techniques. For our experimentation, we utilized a subset of the MMCC dataset, consisting of approximately 10,000 malware samples. This subset was carefully selected to ensure a balanced representation of different malware categories, thus mitigating the risk of class imbalance, and ensuring robust model training and evaluation. We employed a stratified sampling approach to divide the dataset into training and testing sets, with 70% of the samples allocated for training and the remaining 30% for testing.

In terms of experimental setup, we adhered to best practices in machine learning model development. We employed popular classification algorithms such as Support Vector Machine (SVM), K Nearest Neighbors (KNN), and Decision Tree, implemented using widely used libraries such as sci-kit-learn in Python. Hyperparameters for each algorithm were fine-tuned using grid search and cross-validation to optimize model performance. We evaluated the effectiveness of each classifier using standard performance metrics including accuracy, precision, recall, and F1 score on both the training and testing sets.

The proposed methodology divides analysis into various phases to simplify each phase and reduce the interdependency and complexity of analysis to be able to better understand and perform each step with a focus on that phase.

3.1 Exploratory Data Analysis

This was the first and foremost important phase of the analysis in which we analyzed the data to understand what kind of information our data contains, what is the range of value in each data point, how much information is missing in each column, the variance contained by data points and whether the data is biased or unbiased. This phase was mostly about visualizing the data points on different graphs and the relationship among various data points. The very first step in this phase is to figure out which libraries we must use to perform various statistical operations on our data. In Python, we have the below-given libraries for exploratory data analysis:

- NumPy-the fundamental library for scientific calculations.
- Pandas-library for data analysis and its structure.
- Matplotlib-data visualization (graphs, bar charts, pie charts, etc.).
- Seaborn-data visualization.

The flowing chart of the proposed model is shown in Fig. 1.

3.2 Data Preprocessing

Once the data is available in the organized dimensions (i.e., rows and columns, etc.) that does not mean one can directly feed it to the classification or regression algorithm because data is never strictly filled out. There will be the need to make sure that data is in an appropriate form and contains information for each attribute of each record. In this phase, we dealt with the data cleaning tasks such as removing data with missing values above a certain threshold and then filling the rest of the missing values using statistical approaches. And then applying the low variance filters to remove the data having very little variance as that would have a minimal or approximately no impact in predicting the target class. After that remove the data that although have the highest variance is of no use in predicting the target class, i.e., ‘Machine identifier’ and so on.

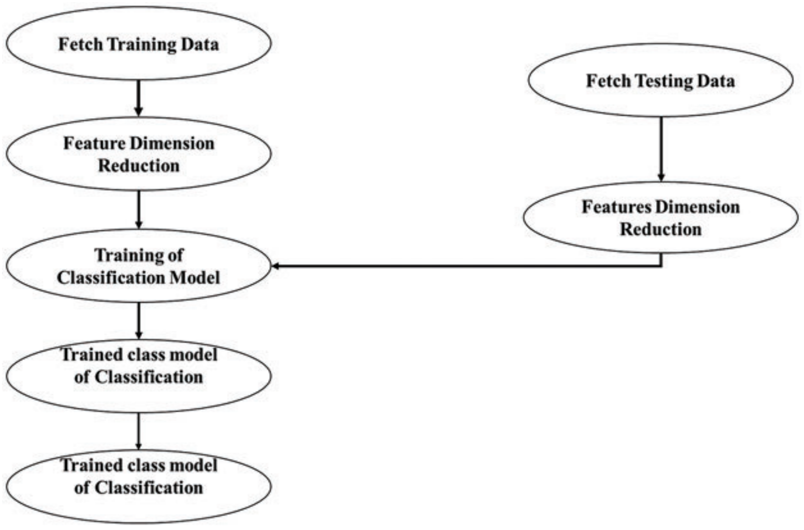
Fetch Training Data
Feature Dimension Reduction
Training of Classification Model
Trained class model of Classification
Trained class model of Classification
Fetch Testing Data
Features Dimension Reduction

Figure 1: Flow chart of the proposed model

3.2.1 Missing Value Ratio

While exploring the data we figured out that there are some missing values in the data. How exactly can we deal with this situation? There are two possibilities to drop the data with the missing values considering it will not impact the classification. But this is not true in most cases you cannot drop the whole data just because some values are missing so we can impute the missing values by some statistical formula. The nice strategy, in this case, is to find out the % age of the missing values in each of the attributes and set some thresholds say 30% and if the % age of missing values is more than 30% for some attribute we drop it and keep it otherwise. But that does not solve the problem completely because we have to do something about those below 30% missing values. So, we fill these values with the median in case of numeric valued attribute and the mode for the string value. Now we have the values filled up we are good to move to the next step.

3.2.2 Low Variance Filter

Now dealing with missing values, that can be seen if the data has the same value for all of the records for a given attribute, then it is not going to affect the classification because the attribute has a zero variance. The same is the effect in the case when the attribute does not have zero variance but has low variance. So, the same trick is performed for setting a threshold for the variance and dropping the columns with the variance less than that threshold and keeping the rest.

3.2.3 High Correlation Filter

The high correlation between the two variables means that similar trends exist between them. For instance, a dependent variable is likely to behave according to the independent variable (y = f(x)) so keeping both variables adds complexity to the model. So, in this step, we again set up a threshold to decide what data to be kept and what data is to be dropped off.

3.2.4 Features Encoding

Although we have improved our data with various techniques so for, there is still one thing that needs to be considered. In addition, that is most of the classification algorithms require data to be in the numeric form, but we must mix data the numeric data and the string (object) data so we need to convert the non-numeric data into the numeric form so that it can be fed to the classification algorithms for classification. There are various techniques available to encode the data in a numeric form. Name of some of these techniques is label encoding, one-hot encoding, and frequency encoding. Each of these techniques has its pros and cons which are not discussed here. We used frequency encoding in Table 3 which replaces each non-numeric value with the frequency of its occurrence in the data. For example, if we have an attribute called 'weather' containing the possible values 'hot, cold, moderate' and we have 100 records in which the value of the 'weather' attribute is as follows:

Table 3: Weather record

Weather	No. of records
Hot	35
Cold	15
Moderate	50
Total	**100**

In frequency encoding, we replace each value with its frequency (i.e., hot with 35, cold with 15, and moderate with 50). Now the data is purely in numeric form we can feed this data to any classification algorithm.

3.2.5 Memory Optimization

Since the real-world data is usually quite large as is the case of our subject data (millions of records), it is not very easy to handle this much data in normal systems with normal capabilities [40]. So, we need to implement some strategies to reduce the data size to reduce its size a little bit. Python keeps the numeric data in int64 and float64 by default but most of the time the data contained by an attribute is much smaller [41]. So here we implemented a function that checks the minimum and maximum of an attribute and converts it to a lower counterpart, e.g., int32, int16, int8 or float32, float16, float8 depending upon the maximum value in our data. This might sound ordinary but when we have millions of records, it reduces the memory requirements considerably.

3.2.6 Features Dimension Reduction

The data has been cleaned, but the data may still be redundant and contain a good mix of attributes contributing the maximum to the prediction of the target class and the attributes that have very little contribution towards the prediction of our target class. Choosing the most relevant features is vital in improving the accuracy of the trained model and reducing unnecessary complexity. For this purpose, we used the famous technique called principal component analysis (PCA) which finds the new axes known as principal components in data based on the variance contained by the data points and leaves the axes with low variance [42]. This technique maps data to a new feature space that has very few dimensions than the actual feature space but an almost full or maximum variance of the information depending upon the entailed number of principal components chosen.

3.3 Model Training

We explore the critical stage of model training in Section 3.3, where the choice and use of classification algorithms are described in detail concerning experimental design. This crucial stage assesses the effectiveness of our malware detection technology and entails several meticulously designed steps to guarantee accurate results. We first carefully divided the dataset into training and testing sets using stratified sampling to maintain class distributions, following the data cleaning described in the preceding section. Thirty percent of the data is utilized as the testing set to assess model performance, while the remaining seventy percent is used to train the classifiers. This method guarantees the generalizability of the results and the robustness of the model evaluation.

Moving on to classifier training, we use a methodical approach, starting with Support Vector Machine (SVM). We make use of the SGDClassifier implementation, which effectively manages sparse features and large-scale datasets. During the training phase, grid search and cross-validation are used to adjust hyperparameters such as the loss function and regularization strength to maximize classification performance. Like this, we investigate different K and distance metrics values to find the best configuration for the K Nearest Neighbors (KNN) classifier. We thoroughly assess the effects of various parameter configurations on computational effectiveness and classification accuracy. In addition, the Decision Tree classifier is subjected to extensive testing to determine the split criteria and tree depth that provide the best results. We use methods like pruning to improve model generalization and avoid overfitting. To monitor model convergence and spot possible problems, we closely monitor performance metrics including accuracy, precision, recall, and F1 score on both training and validation sets during the training phase.

To maximize the power of several classifiers, we additionally investigate ensemble techniques like Random Forest and Gradient Boosting in addition to these main classifiers. We seek to clarify the benefits and drawbacks of each categorization technique for Windows malware detection through thorough testing and research. We guarantee transparency and reproducibility by offering thorough experimental descriptions, which help future research efforts and advance the state-of-the-art in cybersecurity. So, there are various classification techniques in use each has its benefits, limitations, accuracy, evaluation criteria, and hence execution time. But the general steps are the same as follows.

3.3.1 Data Cleaning

The very first step is cleaning the data which we have described in the previous section with details.

3.3.2 Train and Test Split

Then we split the data into two parts Training data and Testing data. Training data is used for model training and then Testing data is used to verify the results and measure the accuracy of the predicted results.

3.3.3 Training

In this step, we apply the algorithm to train the classifier on our data.

3.3.4 Prediction

The classifier is provided with the unseen data to predict the target class from the patterns learned during a training phase.

3.3.5 Accuracy

Then we compare the actual target class and the predicted class to measure the accuracy of the Model. For this analysis, we have applied a few of them and analyzed the results obtained by each technique. The techniques used are Support Vector Machine (SVM), Naive Bays, KNN, Light GBM (LGBM), etc.

3.3.6 Support Vector Machine (SVM)

SVM is a supervised classification technique that is used for classification, regression as well as outlier detection. SVM has several advantages like it works effectively in high dimension data and also having various kernel functions for predicting decision boundaries [43]. Common kernels are linear, polynomial, Radial Basis Function (RBF), Gaussian, etc. SVM has the ins that it chooses the decision boundary which maximizes the distance of the nearest point from the decision boundary from each side as shown in Fig. 2.

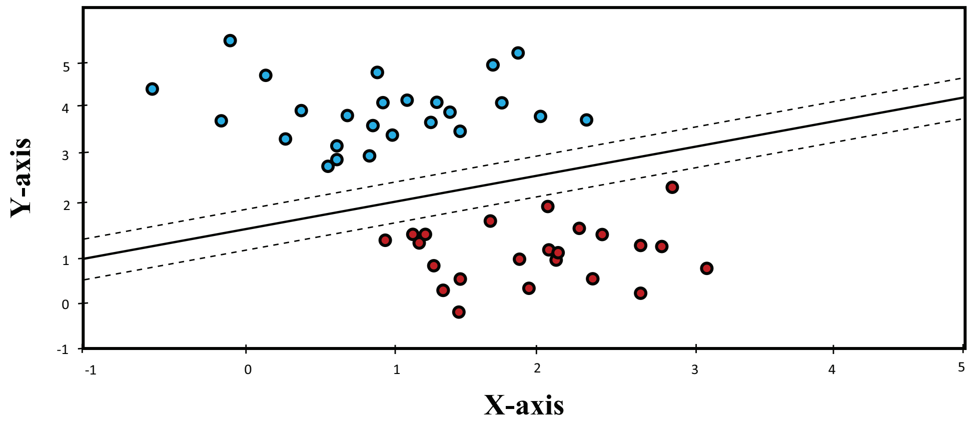

Figure 2: SVM classifier

In this analysis, we have used a special version of SVM called SGDC which is better suited for large-scale and sparse machine learning problems such as text classification, etc. As with simple SVM, there are also hyperparameters that the function takes and the most important one is the 'loss' parameter which decides the decision boundary of the classifier, i.e., whether it will be linear or non-linear, etc.

3.3.7 K Nearest Neighbors (KNN) Classifier

KNN is a versatile and robust algorithm that is easy to understand as well. It is used for classification, regression, and clustering problems. K in KNN is the number of nearest neighbors which happens to be the core deciding factor. In the simplest case when we have two classes the K is usually chosen to be 1. KNN then finds the distance of the point to be predicted from 1 nearest point in the data and predicts the class of that point as presented in Fig. 3.

When using K greater than 1, usually an odd number is chosen as the value of K, KNN calculates the distance of K's closest points from point P and decides the class of point P by the voting of most of the neighbor's class as described in Fig. 4.

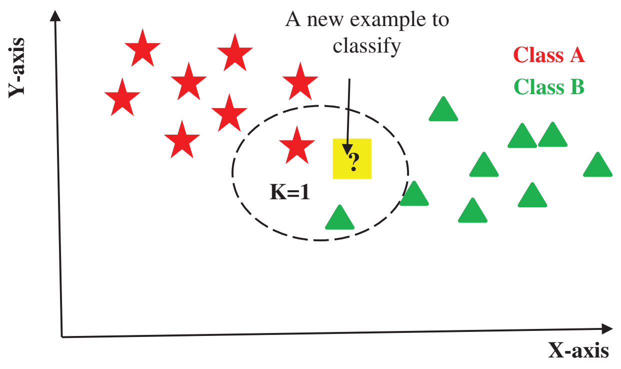

Figure 3: KNN classifier

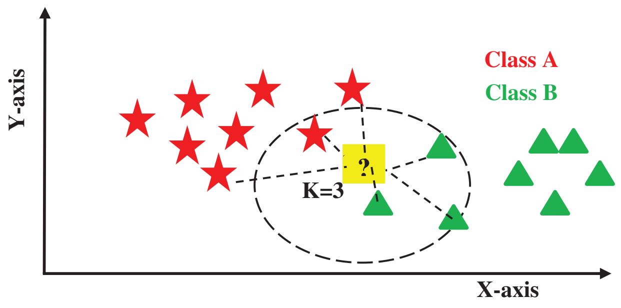

Figure 4: KNN classification for two classes

The question is how to choose the value of K. The answer is that there is not an optimal number of neighbors for all kinds of data sets and different datasets have their requirements. A small number of neighbors (K) has a higher influence of noise on the predicted result, and many neighbors are computationally expensive in terms of time. Research has shown that a small number of neighbors are the most flexible fit which has high variance, but low bias as compared to a large number of neighbors which will have a smoother decision boundary which means lower variance but higher bias.

3.3.8 Decision Tree

A decision tree is one of the most important algorithms in data science and is widely used for classification as well as regression problems. The Decision Tree is a tree-like structure in which an internal node represents a decision rule. The top-most node is the root where the leaf nodes are the outcome. At each node, the decision is made via ASM (Attribute Selection Measure), and the dataset is broken into smaller datasets [44]. A decision tree as shown in Fig. 5 is faster in terms of time complexity as compared to neural networks. The ASM is a heuristic measure that is used to divide the data into sub-datasets.

Besides these classifiers, we also have tried a few more of the classifiers like Naïve Bayes, LGBM, etc., to check whether we get better than these. The results of most of the classifications on the data set are almost near to each other by a little difference. All classifiers yielded results between 50% and 60%. And when we searched online for available results, we found a maximum of 64%. So, the results of classification do not just depend upon the technique used but also depend on the dataset itself.

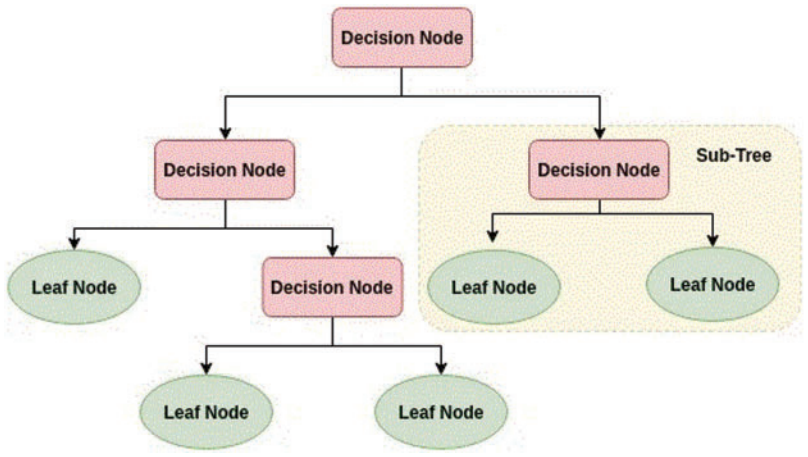

Figure 5: Structure of the decision tree

4 Results and Discussion

This section provides results and discussions of the overall proposed work.

4.1 Results Obtained from SGDC Classifier

Table 4 presents the results obtained from the SGDC classifier, showcasing various attributes such as alpha, epsilon, learning rate, loss, max_iter, and verbose. The testing accuracy of approximately 95.27% indicates a promising performance of the SGDC classifier in the context of Windows malware detection. The selection of optimal hyperparameters, including alpha and epsilon, plays a crucial role in determining the classifier's effectiveness in distinguishing between benign and malicious samples. Moreover, the choice of loss function, whether hinge or otherwise, significantly impacts the decision boundary, thereby influencing the classifier's overall performance. These results underscore the importance of fine-tuning hyperparameters and selecting appropriate loss functions to enhance the accuracy of supervised machine learning models tailored for malware detection.

Table 4: SGDC classifier results

Attribute	Value
Alpha	0.0001
Epsilon	0.1
Learning rate	Optimal
Loss	Hinge
Max_iter	5
Verbose	0

The testing accuracy happened to be around 95.27% yielded by the SDGC classifier with the above parameters shown in Table 4. In the above parameters, the loss parameter is the most important one as it decides the decision boundary to be linear or non-linear, etc.

4.2 Results of KNN with Varying K

We used the KNN algorithm by changing the value of K and the results obtained by each one are shown in Table 5 as given below.

Table 5: KNN results for K = 1

Attribute	Value
Algorithm	Auto
Leaf size	30
Metric	Minkowski
Metric_params	None
Neighbor's	1
Weights	Uniform

4.2.1 Results Obtained from K Nearest Neighbors When K = 1

The accuracy obtained from K Nearest Neighbors when K is taken as 1 is around 93%. In KNN the deciding factor is K which predicts the target class as shown in Table 5.

4.2.2 Results Obtained from K Nearest Neighbors When K = 2

The accuracy obtained from K Nearest Neighbors when K is taken as 1 is around 92% with the above parameters shown in Table 6.

Table 6: KNN results for K = 2

Attribute	Value
Algorithm	Auto
Leaf size	30
Metric	Minkowski
Metric_params	None
Neighbor's	2
Weights	Uniform

4.2.3 Results Obtained from K Nearest Neighbors When K = 3

The accuracy with the value of K = 1 of the K Nearest is approximately around 94%. Note that the other parameters are set to default values, but we are changing the value of K only as it is the main player in predicting the class of unknown points as shown in Table 7.

4.2.4 Results Obtained from K Nearest Neighbors When K = 4

The accuracy with the value of K taken to be 4 KNN classifier is slightly less than that of compared with K = 3, i.e., 95.4%. Other parameters are set to default values, but we are changing the value of K only to predict the target class of unknown point as described in Table 8.

Table 7: KNN results for K = 3

Attribute	Value
Algorithm	Auto
Leaf_size	30
Metric	Minkowski
Metric_params	None
Neighbor's	3
Weights	Uniform

Table 8: KNN results for K = 4

Attribute	Value
Algorithm	Auto
Leaf_size	30
Metric	Minkowski
Metric_params	None
Neighbour's	4
Weights	Uniform

4.2.5 Results Obtained from K Nearest Neighbors When K = 5

The accuracy when K is 5 yields the result of approximately 93.6% We can see as we vary the value of k the output varies as well but there is no clear pattern of whether it is increasing with K or decreasing because there is no specific rule of how we should choose the value of K as shown in Table 9. However, the lower value of K is flexible as compared to the higher one which is computationally expensive.

Table 9: KNN results for K = 5

Attribute	Value
Algorithm	Auto
Leaf_size	30
Metric	Minkowski
Metric_params	None
Neighbor's	5
Weights	Uniform

4.3 Accuracy of the Result with Gaussian Naïve Bayes

We have used a variant of the Naïve Bayes algorithm from the ticket-learn implementation. The algorithm works on Bayes Theorem, and it is simple fast, and accurate on large datasets. The Naïve Bayes works by calculating probabilities of class labels to predict the target class. It applies the Bayes formula to predict the target class. The accuracy score obtained by the Gaussian Naïve

Bayes is approximately 99.54%. Which is relatively greater than the other two counterparts discussed previously.

4.4 Accuracy Result Obtained from Decision Tree

The decision tree with the above-mentioned default parameters yields an accuracy of 97.8% which is slightly less than the accuracy of Gaussian Naïve Bays which was 99.54% with the above parameters shown in Table 10.

Table 10: Decision tree results

Attribute	Value
Class_weight	None
Max_depth	3
Max_leaf_nodes	None
Min_samples_leaf	5
Min_samples_split	2
Presort	False
Random_state	100
Splitter	Best
Criterion	Gini

Table 11 enlists the comparison results of the supervised machine learning classifier for malware detection in Windows machines. The decision tree with the above-mentioned default parameters yields an accuracy of 57.8%, SGDC of 59.2%, and KNN of 53.3%, which is slightly less than the accuracy of Gaussian Naïve Bays which is 59.54%.

Table 11: Comparison of supervised machine learning classifier

Classifier	Accuracy (%)
SGDC	95.2
CNN	93.3
Gaussian Naïve Bayes	99.54
Decision Tree	97.4

Table 12 provides a comparative analysis of our study's results with those of similar research endeavors in the field of Windows malware detection. Our supervised machine learning classifiers demonstrate competitive accuracy rates, with the Gaussian Naïve Bayes model achieving a notable accuracy of 99.54%. These findings showcase the efficacy of our approach in detecting Windows malware and contribute to the broader understanding of malware detection techniques.

4.5 Comparative Analysis of Results

In the pursuit of understanding the efficacy of our chosen supervised machine learning classifiers for Windows malware detection, we compare our findings with existing studies in the field. While similar methodologies and techniques may exist, our study represents a novel contribution, as evidenced by the absence of directly comparable studies in the literature.

Table 12: Results comparison with similar studies

Study	Description	Method	Accuracy (TPR%)
Supervised learning technique	Windows malware	Dynamic	99.54
Elderan [45]	Windows ransomware	Dynamic	96.3
Mobile malware detection [46]	Android malware	Network conversations	96.99
Peershark [47]	P2P Botnets	Network conversations	95.0 (average)

For instance, in a study by Zada et al. [48], a dataset focusing on statistical network traffic aspects was utilized for the analysis of Android malware. Although employing different datasets and platforms, our study shares a common goal of malware detection through supervised machine learning. However, notable differences in methodologies and datasets may account for variations in performance metrics.

Moreover, in the analysis of Windows malware conducted by Wang et al. [49], dynamic analysis was employed to identify malicious features. Their study achieved a True Positive Rate (TPR) of 96.3%, which provides a benchmark for comparison with our findings. While our study may differ in the specific classifiers utilized and the nature of features extracted, such comparative insights contribute to a deeper understanding of the effectiveness of different malware detection approaches.

Similarly, Singh et al. [50] adopted a statistical network conversation approach to analyze botnet traffic, successfully identifying multiple botnet applications with an average TPR of 95.0 %. While their focus differs from our study, which centers on Windows malware detection, the shared emphasis on supervised machine learning underscores the relevance of their findings to our comparative analysis.

Overall, our study adds to the body of knowledge surrounding malware detection by providing a detailed examination of supervised machine learning classifiers tailored for Windows systems. While direct comparisons with existing studies may present challenges due to variations in methodologies and datasets, the insights gleaned from such analyses contribute to a more comprehensive understanding of the landscape of malware detection techniques.

4.6 Discussion

In addition to comparing supervised machine learning-based Windows malware detection methods, it is imperative to address the challenges posed by complex malware variants and adversarial samples. As cyber threats become increasingly sophisticated, malware authors employ techniques such as polymorphism, obfuscation, and evasion to evade detection by traditional security measures. To effectively handle complex malware, researchers and practitioners must explore advanced feature extraction techniques and model architectures capable of capturing intricate patterns and behaviors exhibited by malicious software. This may involve leveraging deep learning approaches, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), which have demonstrated promising results in detecting complex malware variants by learning hierarchical representations of malware features.

Moreover, the rise of adversarial attacks presents a significant concern for supervised machine learning-based malware detection systems. Adversarial samples are specifically crafted to exploit vulnerabilities in machine learning models, leading to misclassification and potentially bypassing detection mechanisms altogether. To address this challenge, researchers are investigating techniques such as adversarial training, defensive distillation, and robust optimization, which aim to enhance the resilience of machine learning models against adversarial manipulation. By incorporating adversarial robustness into the model training process, cybersecurity practitioners can improve the reliability and effectiveness of malware detection systems in adversarial environments.

Furthermore, it is essential to establish robust evaluation frameworks that account for the presence of complex malware and adversarial samples in the testing datasets. This involves augmenting existing benchmark datasets with diverse and realistic malware samples, including polymorphic variants and adversarial examples generated using sophisticated attack algorithms. By evaluating detection models under realistic conditions, researchers can assess their performance in identifying both known and novel malware threats while mitigating the risk of false positives and false negatives. Additionally, ongoing collaboration between academia, industry, and government stakeholders is crucial for sharing knowledge, resources, and best practices in combating complex malware and adversarial attacks, ultimately strengthening the cybersecurity posture of organizations worldwide.

5 Conclusion

The study aimed to enhance malware detection on Windows systems using supervised machine learning classifiers. The study explored various classifiers, including Gaussian Naïve Bayes, K Nearest Neighbors, Stochastic Gradient Descent Classifier (SGDC), and Decision Tree, to assess their efficacy in detecting malicious software. Our findings demonstrate promising results, with Gaussian Naïve Bayes achieving the highest accuracy rate of 99.54%, closely followed by Decision Tree at 97.4%. SGDC exhibited a slightly lower accuracy rate of 95.2%, while K Nearest Neighbors achieved 93.3%. These insights highlight the significance of leveraging supervised machine learning for bolstering cybersecurity measures on Windows platforms.

Moving forward, further research could explore hybrid approaches integrating multiple classifiers or leveraging ensemble learning techniques to enhance detection accuracy. Additionally, the development of specialized features tailored to Windows malware characteristics could further fortify malware detection methodologies. In essence, our study underscores the importance of continual innovation in malware detection strategies to safeguard computer systems against evolving threats. By leveraging supervised machine learning techniques, we aim to contribute to the ongoing efforts to fortify cybersecurity measures and mitigate the impact of malicious software on Windows environments.

6 Study Limitations and Future Work

This study highlighted several shortcomings that need careful consideration. First off, the caliber and representativeness of training data have a major impact on how well-supervised machine learning classifiers perform. The generalizability of our findings and the applicability of our models to real-world settings may be hampered by inadequate or biased datasets. Additionally, the selection of features and preprocessing methods might affect classifier performance, possibly adding bias or noise and lowering the accuracy of malware detection. We acknowledge that further research is necessary to fully comprehend these issues, and we offer focused improvement tactics to increase model performance.

Additionally, conventional detection techniques continue to face difficulties due to the dynamic nature of contemporary malware. Detection strategies must always be adjusted and improved to effectively counter new threats as adversaries change their tactics. Moreover, in resource-constrained situations, practical limitations such as computational resource limitations may impede the implementation and scalability of machine learning models. To tackle these obstacles, one must investigate novel approaches, cooperate with cybersecurity specialists, and create stronger detection frameworks. It is recommended that future research concentrate on sophisticated feature engineering techniques, hybrid learning methodologies, and real-time analytic methods to improve cyber resilience and malware detection capabilities.

Acknowledgement: This study is carried out through the support of the International Islamic University Islamabad Pakistan, University of Peshawar Pakistan, Princess Nourah bint Abdulrahman University Riyadh Saudi Arabia, University of Jeddah, and Princess Nourah bint Abdulrahman University Saudi Arabia.

Funding Statement: This research work is supported by Princess Nourah bint Abdulrahman University Researchers Supporting Project Number (PNURSP2024R411), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Author Contributions: Islam Zada: Conceptualization, Methodology, Writing—Original Draft, Mohammed Naif Alatawi: Data Curation, Investigation, Writing—Review & Editing. Syed Muhammad Saqlain: Formal Analysis, Software, Visualization, Abdullah Alshahrani: Supervision, Project Administration, Funding Acquisition., Adel Alshamran: Conceptualization, Methodology, Writing—Review & Editing, Kanwal Imran: Data Curation, Investigation, Writing—Review & Editing, Hessa Alfraihi: Formal Analysis, Visualization, Project Administration. The authors made significant contributions to the research and development of this study. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data used in this work is available on the machine learning repository.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *J. Comput. Virol.*, vol. 2, no. 1, pp. 67–77, 2006. doi: 10.1007/s11416-006-0012-2.

[2] J. Saxe and K. Berlin, "Deep neural network-based malware detection using two dimensional binary program features," in *2015 10th Int. Conf. Malicious and Unwanted Softw. (MALWARE)*, Fajardo, PR, USA, 2015, pp. 11–20.

[3] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—A state of the art survey," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–48, 2019.

[4] S. Choudhary and M. Vidyarthi, "A simple method for detection of metamorphic malware using dynamic analysis and text mining," *Proc. Comput. Sci.*, vol. 54, pp. 265–270, 2015. doi: 10.1016/j.procs.2015.06.031.

[5] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction selection and fusion for effective malware family classification," in *Proc. of the Sixth ACM Conf. on Data and Appl. Security and Privacy, Association for Computing Machinery*, New York, NY, USA, 2016, pp. 183–194.

[6] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, "Data augmentation-based malware detection using convolutional neural networks," *PeerJ Comput. Sci.*, vol. 7, pp. e346, Jan. 2021. doi: 10.7717/peerj-cs.346.

[7] D. Vasan, M. Alazab, S. Venkatraman, J. Akram, and Z. Qin, "MTHAEL: Cross-architecture IoT malware detection based on neural network advanced ensemble learning," *IEEE Trans. Comput.*, vol. 69, no. 11, pp. 1654–1667, 2020. doi: 10.1109/TC.2020.3015584.

[8] T. M. Kebede, O. Djaneye-Boundjou, B. N. Narayanan, A. Ralescu, and D. Kapp, "Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft Malware classification challenge (BIG 2015) dataset," in *2017 IEEE Nat. Aerospace and Electron. Conf. (NAECON)*, Dayton, OH, USA, Jun. 27–30, 2017, pp. 70–75.

[9] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CW sandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007. doi: 10.1109/MSP.2007.45.

[10] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Hum. Centric Comput. Inf. Sci.*, vol. 8, no. 1, pp. 3, 2018.

[11] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Vancouver, BC, Canada, May 26–31, 2013, pp. 3422–3426.

[12] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," *J. Inf. Secur.*, vol. 5, pp. 56–64, 2014. doi: 10.4236/jis.2014.52006.

[13] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "PE-Miner: Mining structural information to detect malicious executables in realtime," in *12th Int. Symp.*, Saint-Malo, France, RAID, Sep. 2009, pp. 121–141.

[14] Q. Yanchen, Z. Bin, and Z. Weizhe, "Malware classification method based on word vector of bytes and multilayer perception," in *IEEE Int. Conf. on Commun. (ICC)*, Dublin, Ireland, Jun. 2020, pp. 1–6.

[15] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles," *EURASIP J. Inf. Secur.*, vol. 2017, no. 1, 2017. doi: 10.1186/s13635-017-0055-6.

[16] S. Cesare and Y. Xiang, "Classification of malware using structured control flow," in *Proc. of the Eighth Australasian Symp. on Parallel and Distrib. Comput.*, Brisbane, Australia, Jan. 01, 2010, vol. 107, pp. 61–70.

[17] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020. doi: 10.1007/s10462-020-09825-6.

[18] B. Ndibanje, K. H. Kim, Y. J. Kang, H. H. Kim, T. Y. Kim and H. J. Lee, "Cross-method-based analysis and classification of malicious behavior by API calls extraction," *Appl. Sci.*, vol. 9, no. 2, pp. 239, 2019. doi: 10.3390/app9020239.

[19] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli and A. K. Sangaiah, "Classification of ransomware families with machine learning based on *N*-gram of opcodes," *Future Gener. Comput. Syst.*, vol. 90, pp. 211–221, Jan. 2019. doi: 10.1016/j.future.2018.07.052.

[20] W. El-Shafai, I. Almomani, and A. AlKhayer, "Visualized malware multi-classification framework using fine-tuned CNN-based transfer learning models," *Appl. Sci.*, vol. 11, no. 14, pp. 6446, Jul. 2021. doi: 10.3390/app11146446.

[21] A. Bensaoud, J. Kalita, and M. Bensaoud, "A survey of malware detection using deep learning," *Machine Learn. Appl.*, vol. 16, no. 1, pp. 100546, Jun. 2024. doi: 10.1016/j.mlwa.2024.100546.

[22] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: A deep learning framework for intelligent malware detection," *Proc. Int. Conf. Data Mining Steering Committee World Congr. Comput. Sci. (DMIN)*, Las Vegas, NV, USA, Jul. 2016, pp. 61–67.

[23] E. Gandotra, D. Bansal, and S. Sofat, "Tools & techniques for malware analysis and classification," *Int. J. Next-Generation Comput.*, vol. 7, no. 3, pp. 176, 2016.

[24] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Comput. Appl.*, vol. 31, no. 14, pp. 461–472, Feb. 2020. doi: 10.1007/s00521-017-3077-6.

[25] S. Srinivasan and P. Deepalakshmi, "Enhancing the security in cyber-world by detecting the botnets using ensemble classification-based machine learning," *Meas. Sens.*, vol. 25, no. 2665–9174, pp. 100624, Feb. 2023. doi: 10.1016/j.measen.2022.100624.

[26] U. E. H. Tayyab, F. B. Khan, M. H. Durad, A. Khan, and Y. S. Lee, "A survey of the recent trends in deep learning based malware detection," *J. Cybersecurity Privacy*, vol. 2, no. 4, pp. 800–829, Sep. 2022. doi: 10.3390/jcp2040041.

[27] R. Oak, M. Du, D. Yan, H. Takawale, and I. Amit, "Malware detection on highly imbalanced data through sequence modeling," in *12th ACM Workshop on Artif. Intel. and Security*, London, UK, 15 Nov., 2019, pp. 37–48.

[28] B. Cakir and E. Dogdu, "Malware classification using deep learning methods," in *ACMSE '18: Proc. of the ACM Southeast (ACMSE) 2018 Conf.*, Richmond Kentucky, USA, Mar. 29–31, 2018, pp. 1–5.

[29] V. Verma, A. Malik, and I. Batra, "Analyzing and classifying malware types on windows platform using an ensemble machine learning approach," *Int. J. Performability Eng.*, vol. 20, no. 5, pp. 312–318, 2024. doi: 10.23940/ijpe.24.05.p6.312318.

[30] N. Usman *et al.*, "Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics," *Future Gener. Comput. Syst.*, vol. 118, pp. 124–141, May 2021. doi: 10.1016/j.future.2021.01.004.

[31] J. Zhang, "Machine learning with feature selection using principal component analysis for malware detection: A case study," arXiv:1902.03639, 2019.

[32] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *10th Int. Conf. on Cyber Conflict*, Tallinn, Estonia, May 29–Jun. 01, 2018. pp. 371–390.

[33] S. Kumar and Sudhakar, "MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things," *Future Gener. Comput. Syst.*, vol. 125, pp. 334–351, Dec. 2021. doi: 10.1016/j.future.2021.06.029.

[34] S. Seneviratne, R. Shariffdeen, S. Rasnayaka, and N. Kasthuriarachchi, "Self-supervised vision transformers for malware detection," *IEEE Access*, vol. 10, pp. 103121–103135, 2022. doi: 10.1109/ACCESS.2022.3206445.

[35] Z. Zhao, J. Wang, and J. Bai, "Malware detection method based on the control-flow construct feature of software," *IET Inf. Secur.*, vol. 8, pp. 18–24, 2014. doi: 10.1049/iet-ifs.2012.0289.

[36] X. Lu, F. Jiang, X. Zhou, S. Yi, J. Sha and L. Pietro, "ASSCA: API sequence and statistics features combined architecture for malware detection," *Comput. Netw.*, vol. 157, pp. 99–111, Jul. 2019.

[37] Z. Islam *et al.*, "Enhancing IoT-based software defect prediction in analytical data management using war strategy optimization and Kernel ELM," *J. Wireless Netw.*, vol. 29, no. 8, pp. 1–19, 2023.

[38] G. Ramesh and A. Menen, "Automated dynamic approach for detecting ransomware using finite-state machine," *Decis. Support Syst.*, vol. 138, pp. 113400, Nov. 2020. doi: 10.1016/j.dss.2020.113400.

[39] M. S. Akhtar and T. Feng, "Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time," *Symmetry*, vol. 14, no. 11, pp. 2308, 2022. doi: 10.3390/sym14112308.

[40] F. Ullah *et al.*, "Cyber security threats detection in Internet of Things using deep learning approach," *IEEE Access*, vol. 7, pp. 124379–124389, 2019. doi: 10.1109/ACCESS.2019.2937347.

[41] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi and A. P. Sheth, "Machine learning for Internet of Things data analysis: A survey," *Digit. Commun. Netw.*, vol. 4, no. 3, pp. 161–175, 2018. doi: 10.1016/j.dcan.2017.10.002.

[42] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in industrial Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4329–4341, May 2020. doi: 10.1109/JIOT.2019.2963635.

[43] H. Paulheim and R. Meusel, "A decomposition of the outlier detection problem into a set of supervised learning problems," *Mach. Learn.*, vol. 100, no. 2/3, pp. 509–531, 2015. doi: 10.1007/s10994-015-5507-y.

[44] B. Charbuty and A. Abdulazeez, “Classification based on decision tree algorithm for machine learning,” *J. Appl. Sci. Technol. Trends.*, vol. 2, no. 1, pp. 20–28, Mar. 2021. doi: 10.38094/jastt20165.

[45] S. Aurangzeb, R. N. B. Rais, M. Aleem, M. A. Islam, and M. A. Iqbal, “On the classification of microsoft-windows ransomware using hardware profile,” *PeerJ Comput. Sci.*, vol. 7, pp. 361, Feb. 2021. doi: 10.7717/peerj-cs.361.

[46] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen, “A survey on mobile malware detection techniques,” *IEICE Trans. Inf. Syst.*, vol. E103, no. 2, pp. 204–211, 2020. doi: 10.1587/transinf.2019INI0003.

[47] Z. Zhao *et al.*, “ERNN: Error-resilient RNN for encrypted traffic detection towards network-induced phenomena,” *IEEE Transac. Depend. Secure Computing.*, pp. 1–18, Feb. 2023.

[48] I. Zada, I. Khan, T. Rahman, and A. Jameel, “Classification of software failure incidents using SVM,” *The Sciencetech*, vol. 2, no. 3, pp. 1–13, Sep. 2021.

[49] Z. Wang *et al.*, “BugPre: An intelligent software version-to-version bug prediction system using graph convolutional neural networks,” *Complex Intell. Syst.*, vol. 9, no. 4, pp. 3835–3855, Aug. 2023. doi: 10.1007/s40747-022-00848-w.

[50] N. J. Singh, N. Hoque, K. R. Singh, and D. K. Bhattacharyya, “Botnet-based IoT network traffic analysis using deep learning,” *Secur. Privacy.*, vol. 7, no. 2, pp. 355, Mar. 2024. doi: 10.1002/spy2.355.