International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. , No. (Mon-20..)

http://dx.doi.org/10.12785/ijcds/XXXXXX

Impact of Feature Selection Algorithms in Detecting Android Malware Using Machine Learning Over Permissions and API's

Bilal Ahmad Mantoo[1] **and Zafar Ali Khan N**[2]

[1]*Computer Science and Engineering,Presidency University, Bengalore, India*
[2]*Computer Science and Engineering,Presidency University, Bengalore, India*
Received Mon. 20, Revised Mon. 20, Accepted Mon. 20, Published Mon. 20

Abstract: In recent times, an upsurge of highly sophisticated and intricate malware has emerged, becoming one of the most insidious and perilous attack techniques targeting critical information technology infrastructures. Android, the widely anticipated and open-source smartphone operating system has experienced exponential growth. However, this progress has been impeded by the escalating threat of Android malware, which exploits smartphones to carry out malicious acts. Malware employs a plethora of techniques to circumvent detection systems, presenting novel obstacles to reliable detection. Detecting Android malware efficiently and accurately is crucial in ensuring the security of Android OS users. Machine learning techniques have been widely employed to address this problem, and feature selection algorithms have been introduced to enhance the detection process. This paper investigates the impact of feature selection algorithms specifically applied to permission and API method information in detecting Android malware using different machine learning algorithms. Experiments were conducted to compare the performance of feature selection algorithms, focusing on Principal Component Analysis (PCA) feature selection, F-Score, Recursive feature selection, and Stochastic Neighbor Embedding (SNE). The results demonstrate the effectiveness of the PCA algorithm-based approach in selecting relevant features for malware detection, showing advantages over all feature selection algorithms and reducing the model-building time significantly. The findings highlight the importance of feature selection in optimizing the machine learning-based malware detection system. By selecting pertinent features, the detection process becomes more efficient, improving both accuracy and speed. The PCA algorithm-based feature selection approach outperformed the Feature selection method, showcasing its ability to effectively identify features relevant to Android malware detection.

Keywords: Android Malware, F-Score,Recursive Feature (RFE) Elimination,Stochastic Neighbor Embedding (SNE),Principal Component Analysis (PCA).

1. INTRODUCTION

Android is an open-source operating system primarily designed for mobile devices, such as smartphones and tablets. Developed by Google, Android has become one of the most widely used mobile platforms globally. Its flexibility, customizability, and vast ecosystem of apps have contributed to its popularity among both users and developers. Android provides a rich set of features and capabilities, allowing users to perform various tasks, including communication, web browsing, multimedia consumption, gaming, and productivity. It offers a user-friendly interface, seamless integration with Google services, and support for a wide range of hardware devices. One of the key strengths of Android is its app ecosystem. The Google Play Store offers millions of applications that cater to diverse user needs and preferences. From social networking and entertainment to education and productivity, Android apps cover a broad spectrum of categories. This extensive app ecosystem has fueled innovation and transformed the way people interact with their mobile devices.

Android's open-source nature has fostered a vibrant community of developers, contributing to the continuous evolution and improvement of the platform. The Android Open Source Project (AOSP) enables developers to customize and modify the Android source code to create tailored versions of the operating system. Android also offers seamless integration with other Google services, such as Google Maps, Gmail, Google Drive, and Google Assistant, providing a cohesive user experience across different devices and services. In addition to smartphones and tablets, Android has expanded into other domains, including smart TVs, smart watches, smart home devices, and automotive systems. This versatility has positioned Android as a pervasive platform that powers a wide range of connected devices.

E-mail address: bilalbashir136@gmail.com, zafarallikhan@presidencyuniversity.in

Given the widespread use of smartphones and the increasing number of malware threats targeting the Android platform. Traditional signature-based detection methods often struggle to keep up with the rapidly evolving malware landscape, necessitating the adoption of more advanced techniques such as machine learning. Machine learning has emerged as a powerful approach for Android malware detection due to its ability to analyze large volumes of data and learn patterns that differentiate between normal and malicious behaviors. By training machine learning models on labeled datasets containing both benign and malicious samples, it becomes possible to develop robust and effective malware detection systems.

However, there are challenges in Android malware detection using machine learning, including the imbalance between the number of benign and malicious samples, feature selection, and the potential for adversarial attacks that attempt to evade detection. Android malware detection using machine learning offers a promising approach to combat the evolving threat landscape. By leveraging the power of data analysis and pattern recognition, machine learning models can provide effective protection against malware, ensuring the security and privacy of Android devices and their users.

2. LITERATURE SURVEY

Android Permissions Demystified provides a comprehensive analysis of the Android permission system and its implications for security. It discusses the challenges and limitations of the permission system, including the use of overly broad permissions and the potential for user confusion [1].Behavior-Based Malware Detection System for Android highlight the limitations of traditional signature-based approaches and emphasize the importance of analyzing the behavior of applications to identify potential malware. They explain the components of Crowdroid, including the instrumentation framework, system monitoring, and behavior analysis modules [2].The paper begins by addressing the issue of repackaging, where malicious actors modify legitimate applications to introduce malicious behaviors or bypass security checks. The authors propose a novel detection technique called "Kirin," which utilizes a combination of static and dynamic analysis to identify repackaged applications. [3]. Focusing on the problem of detecting Android malware using machine learning algorithms as an effective approach to identify malicious applications [4].

The DREBIN system employs a combination of static and dynamic analysis techniques to detect Android malware. To evaluate the effectiveness of DREBIN, the authors used a large dataset of over 120,000 android applications, the results showed that DREBIN achieved high accuracy in detecting Android malware while providing detailed explanations for the detection decisions [5].The AndroD-Tector approach leverages ensemble learning by training multiple base classifiers on different subsets of the dataset. They compared the detection accuracy of AndroDTector with several other popular machine learning algorithms and observed significant improvements in terms of precision, recall, and F1-score [6].

By considering multiple indicators of malicious behavior, such as permissions and API calls, machine learning-based classification models can effectively distinguish between benign and malicious applications [7]. Effective malware detection approach by leveraging machine learning techniques and focusing on the analysis of permissions requested by Android applications. The paper begins by first extract the permission requests from Android applications [8]. They then use various machine learning algorithms, such as Support Vector Machines (SVM) and Random Forests, to train classification models based on these permissions [9]. Emphasize the importance of considering both the permissions requested by an application and the API calls it makes, as these can provide valuable insights into its behavior and potential malicious activities. In their proposed method, the authors first extract the requested permissions and API calls from Android applications. They then construct feature vectors based on these extracted information. To classify applications as benign or malicious, they utilize machine learning algorithms, such as Support Vector Machines (SVM) and Random Forest, to train and build prediction models using labeled datasets [10].

Highlighting the increasing prevalence of Android malware and the challenges associated with its detection a comprehensive understanding of an application's behavior and identify potential malicious activities is required. To evaluate the effectiveness of their approach, the authors conducted experiments using real-world Android applications [11]. They measured the accuracy, precision, recall, and F1-score of their models to assess the detection performance. The results demonstrated that their method achieved high detection rates and effectively identified malicious applications [12].The analysis of permission patterns has shown promising results in improving Android malware detection accuracy and reliability [13]. Furthermore, the combined analysis of both permissions and API calls significantly enhances detection accuracy compared to using either feature alone [14], [15].

3. Android Malware Classification

Android malware classification is a process that involves analyzing and categorizing Android malware samples based on their characteristics and attributes. The goal is to identify and classify different types of malware to understand their behavior, potential risks, and develop effective countermeasures. There are several approaches and techniques used in Android malware classification.

A. Static Analysis

Static analysis of Android malware detection involves examining the properties and characteristics of Android applications (APK files) without executing the code. It focuses on analyzing the structure, behavior, and other static features of the malware to determine if it exhibits malicious

behavior or poses a security risk. This analysis is performed by automated tools or security researchers to identify and classify malware samples. Here are some key aspects of static analysis in Android malware detection.

1. Manifest File Analysis: The AndroidManifest.xml file provides essential information about the application, including its package name, requested permissions, declared activities, services, and receivers. By analyzing the manifest file, it is possible to identify suspicious permissions, excessive privileges, or unexpected components that may indicate malicious behavior [16].

2. Permission Analysis: Android apps request permissions to access various resources and functionalities of the device. Analyzing the requested permissions can provide insights into the potential behavior of the app [17]. Suspicious or unnecessary permissions, especially those related to sensitive data or system-level operations, may indicate malicious intent.

3. Code Analysis: Static analysis involves examining the actual code of the Android application. This can include analyzing the Java bytecode, decompiling the APK file to obtain the source code, and analyzing the structure and logic of the code [18]. Various static analysis techniques, such as control flow analysis and data flow analysis, can be applied to identify potentially malicious or suspicious code patterns.

4. API Call Analysis: Android applications interact with the device's operating system and other system components through APIs (Application Programming Interfaces). Analyzing the API calls made by the application can reveal its intended functionality and potential malicious activities. Patterns of API calls associated with known malware behaviors or unauthorized operations can be used to detect and classify malware [19].

5. Code Obfuscation Analysis: Malware authors often employ code obfuscation techniques to make the analysis and reverse engineering of their malware more challenging. Static analysis techniques can be used to identify and deobfuscate the code, uncovering the hidden functionalities and potential malicious activities [20].

6. Signature-based Analysis: Signature-based analysis involves comparing the fingerprints or signatures of known malware samples with the analyzed application [21]. This approach relies on maintaining a database of known malware signatures and matching them against the analyzed APK file to identify known threats.

B. Dynamic Analysis

Dynamic analysis in Android malware detection involves monitoring the behavior and activities of an application during runtime. It focuses on observing how the application interacts with the device, accesses resources, communicates with external entities, and executes code [22]. By analyzing the dynamic behavior of an application, it is possible to detect and classify malware based on their malicious activities or deviations from normal behavior. Here are some key aspects of dynamic analysis in Android malware detection.

1. Emulation or Execution: Dynamic analysis often involves running the application in a controlled environment, such as an emulator or a virtual machine, to observe its behavior. This allows for the monitoring of system calls, API calls, network traffic, file operations, and other runtime activities [23].

2. System Call Monitoring: Dynamic analysis tools intercept and monitor system calls made by the application. System calls provide access to various operating system services and resources. By analyzing the sequence and parameters of system calls, it is possible to identify suspicious or unauthorized operations, such as file system modifications, network communication to malicious servers, or attempts to escalate privileges [24][20].

3. API Call Monitoring: Similar to system call monitoring, dynamic analysis tools also monitor API calls made by the application. Android applications interact with various APIs to access system services, hardware functionalities, and external resources. By analyzing the APIs invoked by the application, it is possible to identify potential malicious behaviors or deviations from normal usage patterns [25].

4. Network Traffic Analysis: Dynamic analysis involves capturing and analyzing the network traffic generated by the application. This allows for the detection of communication with suspicious or malicious servers, data exfiltration attempts, or the use of insecure protocols. Network traffic analysis can help identify malware that relies on command-and-control (CC) infrastructure or data exfiltration techniques.

5. Behavioral Analysis: Dynamic analysis involves observing the application's behavior during runtime. This includes monitoring activities such as the creation and deletion of files, access to sensitive data or resources, background processes or services, and interactions with the user interface. Deviations from expected or normal behavior can indicate malicious activities, such as stealthy data collection, unauthorized access, or privacy violations [26].

6. Malware Signature Matching: Dynamic analysis can also involve comparing the observed behavior of the application against known malware signatures or behavioral patterns. This approach helps identify previously identified malware based on their characteristic behavior or specific actions they perform [27].

Dynamic analysis is particularly effective in detecting sophisticated malware that employs evasion techniques, runtime code loading, or behavior that only manifests during execution. However, it can be more resource-intensive and

time-consuming compared to static analysis.

C. Hybrid approach

A hybrid approach in detecting Android malware refers to the combination of multiple detection techniques and strategies to improve the accuracy and effectiveness of malware detection. It leverages both static and dynamic analysis methods, as well as other complementary techniques, to achieve a more comprehensive and robust detection capability. The hybrid approach aims to overcome the limitations and weaknesses of individual detection techniques by leveraging their strengths in a complementary manner. Here are some key components and techniques commonly used in a hybrid approach for Android malware detection [28].

1. Static Analysis: Static analysis involves examining the application's code and resources without executing it. It includes techniques such as examining the permissions requested by the application, analyzing the manifest file, scanning for known malware signatures or patterns, and inspecting the bytecode or source code for potentially malicious behavior. Static analysis helps identify suspicious or potentially malicious applications before they are even installed or executed [22].

2. Dynamic Analysis: Dynamic analysis, as discussed earlier, involves monitoring the behavior and activities of an application during runtime. It focuses on observing how the application interacts with the device, accesses resources, communicates with external entities, and executes code. Dynamic analysis helps detect malware based on their actual behavior, deviations from expected behavior, or the presence of specific malicious activities [29]. It can capture system calls, API calls, network traffic, and other runtime events to identify potential malware behavior.

3. Machine Learning: Machine learning techniques are often employed in a hybrid approach to improve the accuracy of malware detection. Machine learning models are trained on large datasets containing both benign and malicious samples to learn patterns and characteristics of malware [30]. These models can then be used to classify new and unknown applications as either benign or malicious based on their features and behavior. Machine learning algorithms such as decision trees, random forests, support vector machines, and neural networks are commonly used in Android malware detection [31].

4. Behavior-based Analysis: Behavior-based analysis focuses on the dynamic behavior of an application during runtime [32]. It involves monitoring and analyzing the activities and interactions of the application with the device, system, and external entities. By defining a set of expected or normal behaviors and identifying deviations from this baseline, behavior-based analysis can detect and classify malware based on their malicious activities [33]. This approach is particularly effective in detecting novel or zero-day malware that may not have known signatures or patterns.

5. Signature-based Detection: Signature-based detection involves comparing the characteristics or signatures of an application against a database of known malware signatures. This approach is useful for detecting well-known malware strains and variants [29]. Hybrid approaches may incorporate signature-based detection as one of the components to quickly identify and classify known malware, while leveraging other techniques to detect unknown or variant malware [15].

A hybrid approach combines the strengths of these techniques, often using a combination of automated analysis tools, machine learning models, and expert-driven analysis, to enhance the accuracy and efficiency of Android malware detection [33]. By leveraging multiple detection methods, it becomes possible to detect a wider range of malware and improve the overall detection rate while minimizing false positives. Implementing a hybrid approach requires careful consideration of the chosen techniques, their integration, and the optimization of detection algorithms.

4. Experimental Methodology

In this section, we delve into the experimentation conducted to detect android malware through permission analysis. Our exploration encompasses several crucial aspects, including a comprehensive description of the dataset, followed by an examination of feature extraction techniques, the process of feature selection, and, finally, the implementation of machine learning methodologies. To commence, we provide an in-depth overview of the dataset employed in our analysis, shedding light on its composition, structure, and relevant characteristics. This allows us to gain valuable insights into the underlying data and establish a solid foundation for subsequent analyses.

Next, we delve into the intricate realm of feature extraction, wherein we extract pertinent attributes and characteristics from the dataset. By employing rigorous evaluation methodologies, we aim to identify the subset of features that best contribute to the accurate detection and classification of Android malware. Finally, armed with a refined set of features, we proceed to implement state-of-the-art machine learning techniques. These powerful algorithms, carefully selected and fine-tuned for our specific task, enable us to train robust models capable of accurately detecting and classifying Android malware. Through iterative experimentation and model refinement, we strive to achieve optimal performance and reliability in our malware detection system. By thoroughly investigating each of these critical components, we aim to provide a comprehensive understanding of the experimentation process undertaken in our pursuit of effective Android malware detection based on permission analysis.

A. Data Set Acquisition

The dataset utilized in this study is the CICAndMal2017 Android malware dataset, which comprises a total of 10,854 applications. Among these, 4,354 applications are classified as malware, while the remaining 6,500 are benign applications. The dataset is sourced from diverse origins,

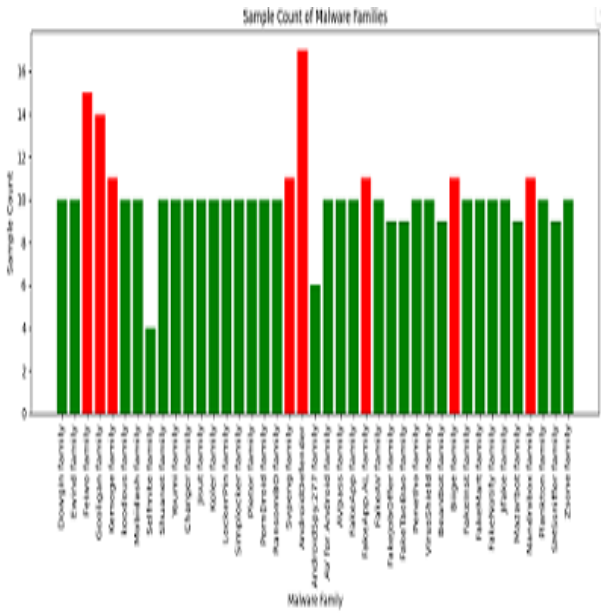

Figure 1. Malware Families from different Sources used in the dataset

ensuring its representativeness and inclusivity. The dataset is organized into four distinct categories, namely Adware, Ransomware, Scareware, and SMS Malware [16] shown in the figure 1. Each category encompasses specific types of Android malware, allowing for a comprehensive analysis of different malware variants.

To ensure diversity and breadth in the analysis, multiple families of datasets have been considered. From each category, a selection of 10-15 families has been included in the study. However, it is noteworthy that only 17 families have been specifically chosen from the swpeng family, as demonstrated in the figure 1. This selective approach enables a focused investigation while still capturing a representative sample of each category, thereby facilitating a comprehensive understanding of the Android malware landscape.

B. Feature Extraction

In this section, we will explore the process of extracting features from the dataset. As our dataset is quite large, extracting and selecting the most suitable features from it poses a significant challenge. To address this, we opted to install a feature extraction library specifically designed for the Android platform. The library we utilized is called Androguard, and it can be easily installed using the pip command in Python. We organized all the applications into a single folder and employed a Python script to extract 216 static features from these applications. The algorithm employed for feature extraction is outlined below algorithm.

The algorithm outlined above provides a comprehensive insight into the intricate process of feature extraction. Initially, all the application features are consolidated within a single folder, and through the utilization of the 'get()' method, these diverse features are diligently accumulated into a python list. Subsequently, this amalgamation of features is diligently preserved as a structured CSV file, ensuring the accessibility and organization of the valuable data.Once the feature compilation is successfully achieved, a transformative step ensues, wherein a binary paradigm is ingeniously applied. This entails the replacement of feature occurrences with either the binary representation of 1 or 0, signifying their presence or absence within the applications.

C. Feature Selection

The aim of feature selection techniques in machine learning is to find the best set of features that allow us to come up with optimized model. In this section we will go through different feature selection techniques and their results and then compare their results to see which one is giving the best possible results.

1) Feature Selection Using F-Score

F-value feature selection, also known as ANOVA (Analysis of Variance), is a statistical technique used to assess the significance of the relationship between a feature and the target variable in a dataset [34]. It helps identify the features that are most relevant or influential in predicting the target variable [35]. Features selected using this algorithm is shown in figure 1.

To assess the performance of the feature selection algorithms, we created charts comparing the accuracy and F1 score obtained by each feature selection method.figure 4, figure5 depicts graphs representing the top ten Features along with their mean F-Value respectively and thier powerBi visual figure 6. These charts figure4 provide a visual comparison of the performance achieved by each method of feature selection. Method:

$$\theta(w,\varepsilon) = 1/2\,\|\omega\|^2 + \sum_{e=1}^{n} \varepsilon \tag{1}$$

subject to: $y_{i(w,x_i+b)} \geq 1 - \varepsilon_i, \varepsilon_i \geq 0$

Where i is corresponding to the target variable.The optimization problem is solved in a dual form.

$$w(\alpha) = \sum_{i=1}^{m} \alpha_i - 1/2 \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j x_i . x_j$$

subject to: 1) $0 \leq \alpha_i \leq C, i = 1...m$

$$2)\ \sum_{x=i}^{m} \alpha_i y_i = 0 \tag{2}$$

$$W = \sum_{k_e S V} y_k a_k x_k \tag{3}$$

$$w_i = 1/2\alpha^7 K\alpha - 1/2\alpha^7 K(-i)\alpha \tag{4}$$

The mathematical model of RFE involves iteratively fitting the estimator E on different feature subsets and evaluating the importance of each feature. At each iteration, the least important feature is eliminated, and the process continues until the desired number of features (k) is reached.RFE is a heuristic algorithm that aims to find

the subset of features that maximizes the performance of the chosen estimator. It reduces the dimensionality of the dataset by eliminating less relevant features and selecting the most informative ones for the given task Fig below shows the top ten features selected using RFE and fig shows the correlational heatmap for the selected features.

D. Feature Selection using Stochastic Neighbor Embedding (SNE)

SNE (Stochastic Neighbor Embedding) is a dimensionality reduction technique used for data visualization. SNE aims to capture the local relationships between data points in high-dimensional space and represent them in a lower-dimensional space while preserving these relationships as much as possible [36].

SNE starts by constructing a probability distribution over pairs of high-dimensional data points, where nearby points have a higher probability of being chosen as neighbors [37]. It then creates a similar probability distribution in the low-dimensional space. The goal is to minimize the difference between these two distributions by optimizing the positions of the low-dimensional points.

The algorithm uses a stochastic approach, where it iteratively adjusts the positions of the low-dimensional points to minimize the divergence between the high-dimensional and low-dimensional probability distributions. This adjustment process is guided by the perplexity parameter, which controls the balance between preserving local and global structure in the visualization [38].

SNE is particularly useful for visualizing complex, high-dimensional data in two or three dimensions, allowing analysts to gain insights into the underlying structure of the data . It has been widely used in various domains, including natural language processing, bioinformatics, and computer vision.

Method: $C = KL(P||Q) = \sum_i \sum_j P_{ij} log \frac{P_{ij}}{q_{ij}}$

Where

$$q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_k - y_l||^2)} \quad (5)$$

E. Feature Selection Using Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction and data visualization [39]. It aims to identify the most important patterns or features in a dataset and represent them in a lower-dimensional space.

The key idea behind PCA is to transform a high-dimensional dataset into a new coordinate system, where the new axes, known as principal components, are orthogonal (uncorrelated) and ranked based on the amount of variance they explain in the original data. The first principal component explains the maximum amount of variance, followed by the second principal component, and so on [40].

Here are the steps involved in performing PCA:

1. Standardize the data: If the variables in your dataset are measured on different scales, it is necessary to standardize them (subtract the mean and divide by the standard deviation) so that they have comparable units [41].

2. Compute the covariance matrix: Calculate the covariance matrix of the standardized data. The covariance matrix shows how each variable in the dataset varies with every other variable.

3. Compute the eigenvectors and eigenvalues: Find the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the directions or principal components of the data, while the eigenvalues indicate the amount of variance explained by each principal component [42].

4. Select the principal components: Sort the eigenvalues in descending order and choose the top k eigenvectors (principal components) that correspond to the largest eigenvalues. These principal components capture the most important information in the data.

5. Project the data: Transform the original data onto the new coordinate system formed by the selected principal components. This involves multiplying the standardized data by the eigenvectors of the chosen principal components.

PCA is widely used in various fields, such as data analysis, machine learning, image processing, and genetics. It can help in reducing the dimensionality of a dataset, visualizing high-dimensional data, removing noise, and identifying important features or patterns [43].

Statistics behind Principal Component Analysis:

While diving deep into the PCA, we should have knowledge about statistics used in the principal component analysis, like standard deviation,variance, eigen values and eigen vectors

Where X is arithmetic mean,n is the number of observations

Variance: Another measure of the spread of data is variance. It is defined as the mean of deviations of each term from its arithmetic mean of whole data.

$$Var(X) = \sum_{i=1}^{n} ((X_i - X')^2/(n-1)) \quad (8)$$

Here X is the arithmetic mean, X-X' is the deviation from the arithmetic mean, and n is the number of observations. Covariance : Covariance is always measured between two dimensions, if we have to measure between variables A, B, C we have to measure between A and B, then B and C, then C and A. The formula for covariance is very similar to variance as shown in eq.9.

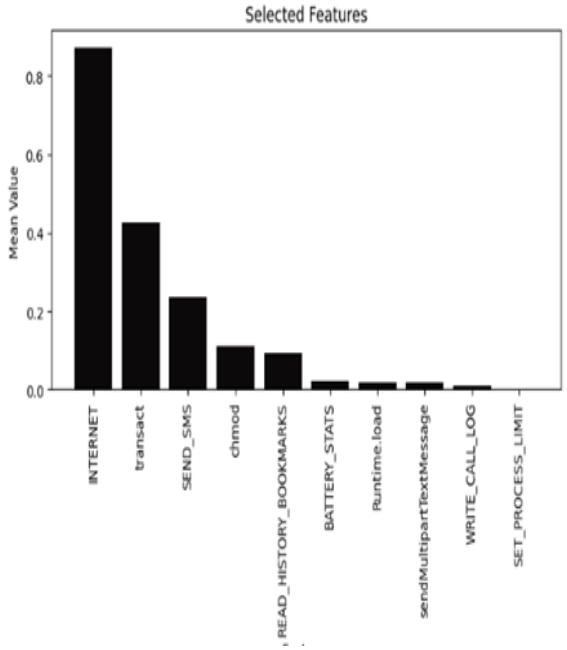

Figure 2. Top Ten Features Using Recursive Feature Elimination as Feature Selection Method

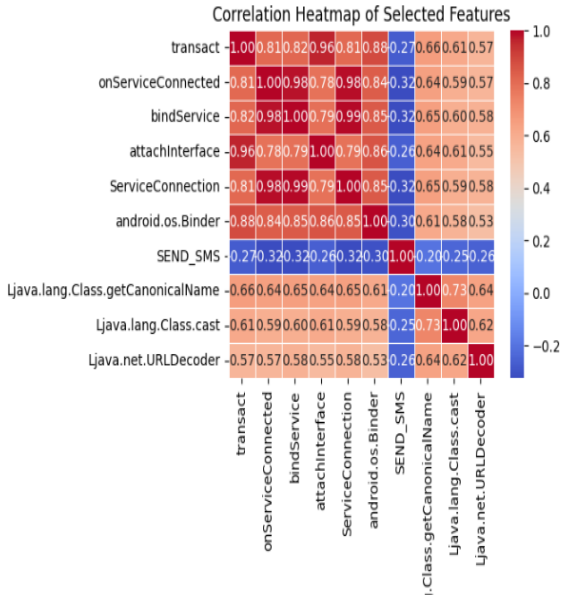

Figure 3. Corelational Heatmap for top ten features using Recursive Feature Elimination

$$\text{Cov(X,Y)}=\sum_{i=1}^{n}(frac(X_i - X^{'})(Y_i - Y^{'})/(n-1)) \quad (9)$$

Here X' is the arithmetic mean of data Y' is the arithmetic mean of data Y.But while dealing with larger space,a useful way is to find covariance between all variables and put them in a matrix as shown

$$^{M*N} = (c_{ij}), c_{ij} = cov(Dim_i, Dim_j) \quad (10)$$

Where C[M*N] is the matrix of M rows and N columns and Dim[i,j] is the ith and jth dimensions.

Similarly we can frind the Eigen value and Eigen vectors using eq 11 and 12 which contain the most important information for feature selection

$$\text{Au}=\lambda u \quad (11)$$

$$(\text{A-}\lambda I)u = 0 \quad (12)$$

Where lambda is a scalar called eigen value associated with eigen vector.

F. Machine Learning

In binary classification machine learning, the performance of the model can be assessed using a confusion matrix, as illustrated in Table 3. This matrix provides valuable information by comparing the predicted classification results with the actual classification results. The confusion matrix is typically presented in a tabular format with four cells, as shown below.

1) True Positive (TP): This represents the number of instances that were correctly predicted as positive, meaning they were accurately classified as the positive class.

2) False Negative (FN): This refers to the number of instances that were incorrectly predicted as negative, meaning they were incorrectly classified as the negative class instead of being positive.

3) False Positive (FP): This indicates the number of instances that were incorrectly predicted as positive, meaning they were mistakenly classified as the positive class instead of being negative.

4) True Negative (TN): This represents the number of instances that were correctly predicted as negative, meaning they were accurately classified as the negative class. By analyzing the values within the confusion matrix, various evaluation metrics can be derived to assess the model's performance. Some commonly used metrics include accuracy, precision, recall (sensitivity), specificity (true negative rate), and F1-score.Here's a brief explanation of these evaluation metrics: - Accuracy: This metric provides an overall measure of the model's correctness, calculated as

$$(TP + TN)/(TP + FN + FP + TN) \quad (13)$$

- Precision: This metric quantifies the proportion of correctly predicted positive instances out of all instances predicted as positive, calculated as

$$TP/(TP + FP) \quad (14)$$

- Recall (Sensitivity or True Positive Rate): This metric measures the proportion of correctly predicted positive instances out of all actual positive instances, calculated as

$$TP/(TP + FN) \quad (15)$$

- Specificity (True Negative Rate): This metric gauges the proportion of correctly predicted negative instances out of all actual negative instances, calculated as.

$$TN/(TN + FP) \quad (16)$$

- F1-score: This metric is the harmonic mean of precision and recall, providing a balance between the two metrics and taking their overall performance into account. It is calculated as.

$$2 * (Precision * recall)/(Precision + Recall) \quad (17)$$

By analyzing the confusion matrix and these evaluation metrics, we can gain insights into the model's performance, identify any issues such as false positives or false negatives, and make informed decisions for further model improvements or adjustments.

5. Results and Discussions

The classification results of the experiment outlined in Section 4 are presented in table 1, which summarizes the accuracy and F1 score of three different methods. The first method involves classification with F-Score as feature selection, using the complete feature set. In this case pipeline method which combines random forest and Knn, performs best as compared to the individual algorithms. The second method incorporates feature selection based on recursive feature elimination (RFE) as shown in table 2, while the third method utilizes feature selection through a Principal Component Analysis (PCA) table 4.

Based on the experimental results, the optimal algorithm for detecting malicious applications varies depending on the method of feature selection. Specifically, SMO performs best without feature selection, IBk performs best with information gain-based feature selection, and Multilayer Perceptron performs best with feature selection using a genetic algorithm. In contrast, NaiveBayes's performance was consistently the lowest across all three cases.

1) Feature Selection Using Recursive Feature Elimination

Recursive Feature Elimination (RFE) is a feature selection method used to select the most relevant features from a given dataset. It is based on the idea of recursively eliminating less important features until the desired number

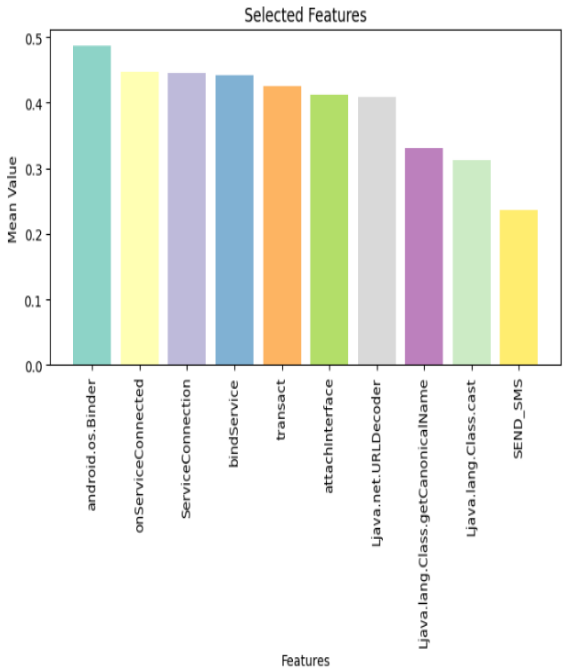

Figure 4. Top Ten Features in a Data set Using their F-Value

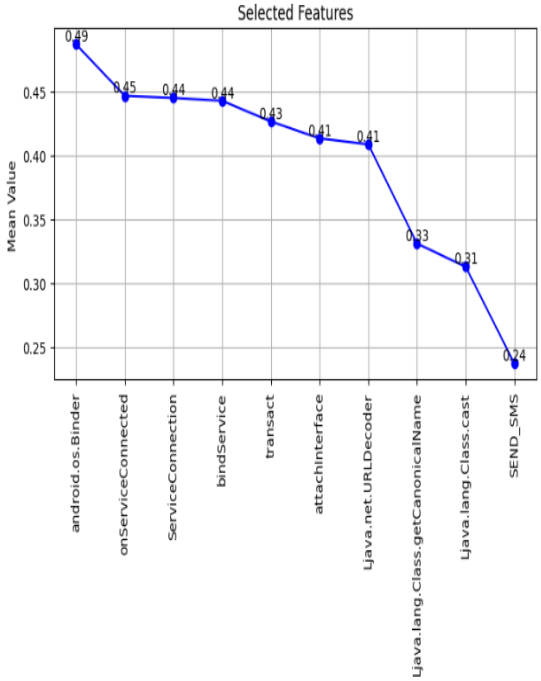

Figure 5. Top Ten Features Using Their Mean F-Value

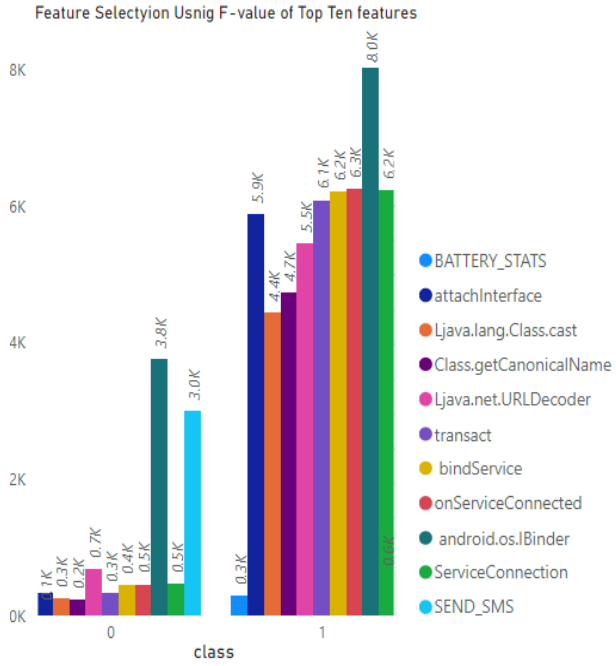

Figure 6. Mean value of Top Ten Features Using Their F-Value

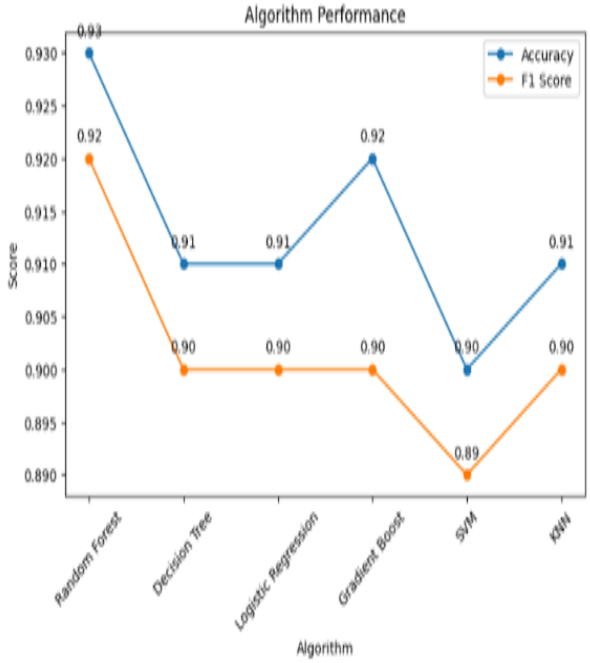

Figure 7. Performance of all the algorithm without pipeline method

of features is reached [44]. The Recursive Feature Elimination (RFE) algorithm can be described mathematically as follows. Let X be the input dataset with n features (X = [x1, x2, ..., xn]) and y be the corresponding target variable [45].

Table 1, shows the result of individual algorithms along with the combination of two algorithms in pipeline Random forest and knn uisng F-Score feature selction.The results shows the pipeline method improves the accuracy of the model.Next feature selection method used is RFE ,which works better than f-value method table 2.Third ,it is also being observed that t-SNE is used as dimensionality reduction tool ,which also helps in increasing the model accuracy table 3.After applying PCA ,the results are not so good,becuase it works better with unsupervised machine learnig table 5,but works better than all other feature selection algorithms used in this work.

Based on the experimental results, the optimal algorithm for detecting malicious applications varies depending on the method of feature selection. Specifically, Random Forest performs best without feature selection, Decision tree performs best with recursive feature elimination, and boost performs best with feature selection using a PCA. In contrast, Knn's performance was consistently the lowest across all three cases.

To assess the performance of the feature selection algorithms, we created charts comparing the accuracy and F1 score obtained by each feature selection method. Figures 7 depict graphs representing the accuracy and F1 score indicators, respectively. These charts provide a visual comparison of the performance achieved by each method of feature selection.

The results from the analysis indicate that the method without feature selection (non-selection) generally outperformed the methods with feature selection (F-value, RFE,t-SNE,PCA). However, the non-selection results only showed small differences of less than 0.04 compared to all machine learning algorithms figure 7. This observation suggests that while feature selection techniques may eliminate a significant number of features, the degradation in performance is relatively low, at only 4 percentage points.

Interestingly, the logistic algorithm showed an increase in accuracy when feature selection was applied. This indicates that the feature selection process was able to identify relevant features for the logistic algorithm, resulting in improved performance. Furthermore, the findings in Figure 6 demonstrate that the F1 scores exhibited similar trends to the accuracy results. The F1 scores of the Gradient Boost algorithm with feature selection (PCA) were higher than those of the method without feature selection (non-selection). This suggests that the feature selection techniques were able to enhance the performance of the Random Forest algorithm by identifying informative features. These insights highlight the benefits of feature selection in reducing the dimensionality of the dataset without significantly sacrificing performance.

TABLE I. Estimated accuracy for algorithms based on multiple feature selection algorithms

Algorithm	Feature Selection Method	Accuracy	F1 Score
Random Forest	F-Value	0.86	0.85
Decision Tree	F-Value	0.95	0.93
Logistic Regression	F-Value	0.94	0.93
Gradient Descent	F-Value	0.93	0.92
Support Vector Machine	F-Value	0.94	0.93
K-Nearest Neighbor	F-Value	0.92	0.91
Pipeline(RF,KNN)	F-Value	0.94	0.93
Random Forest	Principal Component Analysis	0.91	0.90
Decision Tree	Principal Component Analysis	0.91	0.90
Logistic Regression	Principal Component Analysis	0.90	0.89
Gradient Descent	Principal Component Analysis	0.92	0.92
Support Vector Machine	Principal Component Analysis	0.90	0.89
K-Nearest Neighbor	Principal Component Analysis	0.91	0.90
Pipeline(RF,KNN)	Principal Component Analysis	0.93	0.92
Random Forest	Recursive Feature Elimination	0.85	0.84
Decision Tree	Recursive Feature Elimination	0.86	0.85
Logistic Regression	Recursive Feature Elimination	0.82	0.81
Gradient Descent	Recursive Feature Elimination	0.83	0.82
Support Vector Machine	Recursive Feature Elimination	0.83	0.82
K-Nearest Neighbor	Recursive Feature Elimination	0.84	0.82
Pipeline(RF,KNN)	Recursive Feature Elimination	0.92	0.91

A. Time Cost for Model Building

The analysis of accuracy and F1 scores indicates that there is an advantage in not proceeding with feature selection. However, it is important to consider the time required for model building in the machine learning process. Building a model with all features can be time-consuming as it involves learning information from numerous features. Therefore, it becomes necessary to compare the time cost associated with model building Table II.

In some cases, using all features may not be beneficial due to the increased time required. Feature selection techniques can help mitigate this issue by reducing the dimensionality of the dataset, allowing for faster model building. By eliminating irrelevant or redundant features, the model can focus on learning from the most informative features, resulting in reduced training time. Hence, in addition to accuracy and F1 scores, considering the time cost for model building is crucial in determining the optimal approach for the machine learning process. Balancing the trade-off between feature selection, performance, and training time is essential to make informed decisions and achieve efficient and effective machine learning outcomes.

In summary, the results in Table 1 indicate that feature selection plays a crucial role in reducing the time required for model learning. Principal Component Analysis, in particular, demonstrates efficiency in terms of time consumption compared to non-selected data and other feature selection methods. These insights highlight the importance of considering both performance metrics and the time cost associated with feature selection when making decisions in the machine learning process.

6. Conclusions and Future Work

Efficient and accurate detection of Android malware is of utmost importance for users of the Android OS. To address this challenge, numerous studies have explored the application of machine learning techniques for malicious app detection, alongside feature selection methods to expedite the process. In our study, we conducted experiments focusing on selecting permission and API method information features, building upon existing research. The results demonstrate that feature selection using a principle component algorithm proved to be effective compared to the commonly applied algorithm.

Despite a minor reduction in feature selection performance of less than 3 percentage points (p) on average, the pca algorithm-based approach still offers significant advantages over non-selection. By employing the PCA, the model building time is significantly reduced, making the entire detection process more efficient. These findings underscore the importance of feature selection techniques in optimizing the machine learning-based malware detection system. The use of PCA not only ensures a quick and accurate identification of Android malware but also contributes to reducing the computational overhead, enabling a more streamlined and effective detection mechanism.

References

[1] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

TABLE II. Time Cost for Building the model for each algorithm

Algorithm	F-Score	RFE	PCA	
Random Forest	3.32	7.85	7.63	2.36
Decision Tree	0.05	0.01	0.01	0.01
Logistic Regression	0.05	0.01	0.02	0.13
Gradient Descent	35.33	15.19	7.93	6.45
Support Vector Machine	11.18	13.33	5.98	4.32
Knn	13.56	7.95	4.89	3.96
Pipeline(RF,Knn)	9.74	10.14	8.11	8.08

[2] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, pp. 15–26.

[3] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," in *Proceedings of the second ACM conference on Data and Application Security and Privacy*, 2012, pp. 317–326.

[4] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, "De-lady: Deep learning based android malware detection using dynamic features." *J. Internet Serv. Inf. Secur.*, vol. 11, no. 2, pp. 34–45, 2021.

[5] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[6] A. Abusitta, M. Q. Li, and B. C. Fung, "Malware classification and composition analysis: A survey of recent developments," *Journal of Information Security and Applications*, vol. 59, p. 102828, 2021.

[7] C. Zhao, C. Wang, and W. Zheng, *Android Malware Detection Based on Sensitive Permissions and APIs*, 06 2019, pp. 105–113.

[8] W. Z. Zarni Aung, "Permission-based android malware detection," *International Journal of Scientific & Technology Research*, vol. 2, no. 3, pp. 228–234, 2013.

[9] Z. Aung and W. Zaw, "Permission-based android malware detection," *International Journal of Scientific and Technology Research*, vol. 2, pp. 228–234, 01 2013.

[10] X. Xiao and S. Yang, "An image-inspired and cnn-based android malware detection approach," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1259–1261.

[11] N. Rawat, A. Singh, and A. ., "Permission-based malware detection in android using machine learning," *Ymer*, vol. 22(2023), pp. 1505–1517, 03 2023.

[12] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damaševičius, and R. Maskeliunas, "Android malware detection: A survey," 10 2018.

[13] P. Xiong, X. Wang, W. Niu, T. Zhu, and G. Li, "Android malware detection with contrasting permission patterns," *Communications, China*, vol. 11, pp. 1–14, 08 2014.

[14] A. Taha and O. Mohammed, "Intelligent hybrid approach for android malware detection based on permissions and api calls," *International Journal of Advanced Computer Science and Applications*, vol. 8, 01 2017.

[15] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *2012 Seventh Asia joint conference on information security*. IEEE, 2012, pp. 62–69.

[16] D. Wang, T. Chen, Z. Zhang, and N. Zhang, "A survey of android malware detection based on deep learning," in *International Conference on Machine Learning for Cyber Security*. Springer, 2022, pp. 228–242.

[17] K. A. Talha, D. I. Alper, and C. Aydin, "Apk auditor: Permission-based android malware detection system," *Digital Investigation*, vol. 13, pp. 1–14, 2015.

[18] H.-S. Ham and M.-J. Choi, "Analysis of android malware detection performance using machine learning classifiers," in *2013 international conference on ICT Convergence (ICTC)*. Ieee, 2013, pp. 490–495.

[19] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. PP, pp. 1–1, 05 2019.

[20] Q. Li and X. Li, "Android malware detection based on static analysis of characteristic tree," 09 2015, pp. 84–91.

[21] Z.-U. Rehman, S. N. Khan, K. Muhammad, J. W. Lee, Z. Lv, S. W. Baik, P. A. Shah, K. Awan, and I. Mehmood, "Machine learning-assisted signature and heuristic-based detection of malwares in android devices," *Computers & Electrical Engineering*, vol. 69, pp. 828–841, 2018.

[22] B. A. Mantoo, "A hybrid approach with intrinsic feature-based android malware detection using lda and machine learning," in *Recent Innovations in Computing: Proceedings of ICRIC 2020*. Springer, 2021, pp. 295–306.

[23] U. .N, S. Anandaselvi, and D. T Subbulakshmi, "Dynamic malware analysis using machine learning algorithm," 12 2017, pp. 795–800.

[24] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen, "A survey on mobile malware detection techniques," *IEICE Transactions on Information and Systems*, vol. E103-D, pp. 204–211, 02 2020.

[25] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[26] T. Parlar and G. Çınarer, "Feature selection methods for intrusion detection using machine learning methods," vol. 21, no. 7, pp. 63–68, 09 2022.

[27] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, pp. 389–422, 2002.

[28] B. A. Mantoo and S. S. Khurana, "Static, dynamic and intrinsic features based android malware detection using machine learning," in *Proceedings of ICRIC 2019: Recent Innovations in Computing*. Springer, 2020, pp. 31–45.

[29] H.-j. Zhu, W. Gu, L.-m. Wang, Z.-c. Xu, and V. S. Sheng, "Android malware detection based on multi-head squeeze-and-excitation residual network," *Expert Systems with Applications*, vol. 212, p. 118705, 2023.

[30] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE transactions on dependable and secure computing*, vol. 16, no. 4, pp. 711–724, 2017.

[31] N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "A deep dive inside drebin: An explorative analysis beyond android malware detection scores," *ACM Transactions on Privacy and Security*, vol. 25, no. 2, pp. 1–28, 2022.

[32] A. Naway and Y. Li, "Android malware detection using autoencoder," *arXiv preprint arXiv:1901.07315*, 2019.

[33] M. N. AlJarrah, Q. M. Yaseen, and A. M. Mustafa, "A context-aware android malware detection approach using machine learning," *Information*, vol. 13, no. 12, p. 563, 2022.

[34] S. K. Trivedi, "A study on credit scoring modeling with different feature selection and machine learning approaches," *Technology in Society*, vol. 63, p. 101413, 2020.

[35] G. Kumar and K. Kumar, "An information theoretic approach for feature selection," *Security and Communication Networks*, vol. 5, no. 2, pp. 178–185, 2012.

[36] G. E. Hinton and S. Roweis, "Stochastic neighbor embedding," *Advances in neural information processing systems*, vol. 15, 2002.

[37] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova, "Hierarchical stochastic neighbor embedding," in *Computer Graphics Forum*, vol. 35, no. 3. Wiley Online Library, 2016, pp. 21–30.

[38] Z. Yang, I. King, Z. Xu, and E. Oja, "Heavy-tailed symmetric stochastic neighbor embedding," *Advances in neural information processing systems*, vol. 22, 2009.

[39] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[40] M. Ringnér, "What is principal component analysis?" *Nature biotechnology*, vol. 26, no. 3, pp. 303–304, 2008.

[41] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[42] X. XIE, "Principal component analysis," *Wiley interdisciplinary reviews*, 2019.

[43] F. Kherif and A. Latypova, "Principal component analysis," in *Machine Learning*. Elsevier, 2020, pp. 209–225.

[44] K. Yan and D. Zhang, "Feature selection and analysis on correlated gas sensor data with recursive feature elimination," *Sensors and Actuators B: Chemical*, vol. 212, pp. 353–363, 2015.

[45] X.-w. Chen and J. C. Jeong, "Enhanced recursive feature elimination," in *Sixth international conference on machine learning and applications (ICMLA 2007)*. IEEE, 2007, pp. 429–435.

Author 1 Name short biography . . . . . .

Author 2 Name short biography . . . . . .

Author 3 Name short biography . . . . . .