Mesopotamian journal of Cybersecurity
Vol.5,No.2, **pp**. 563–576
DOI: https://doi.org/10.58496/MJCS/2025/034; ISSN: 2958-6542
https://mesopotamian.press/journals/index.php/cybersecurity

Research Article

An Effective Feature Optimization Model for Android Malware Detection

Hussein K. Almulla [1], Hussam J. Mohammed [2,*], Nathan Clarke [3], Ahmed Adnan Hadi [4], Mazin Abed Mohammed [2]

[1] *Computer Center, University of Anbar, Anbar, Iraq.*
[2] *Department of Artificial Intelligence - College of Computer Science and Information Technology, University of Anbar, Anbar, Iraq.*
[3] *School of Engineering, Computing & Mathematics, University of Plymouth, UK.*
[4] *Intelligent Medical Systems Department, College of Sciences, Al-Mustaqbal University, Babil, Iraq.*

ARTICLE INFO

Article history
Received 09 Jan 2025
Revised 11 Apr 2025
Accepted 28 Apr 2025
Published 26 Jun 2025

Keywords
Malware detection
Classification
Feature selection
Ensemble model

ABSTRACT

The rapid development of technology using Android-based smartphones has led to various threats of malware targeting these devices. Over time, android malware has become increasingly complex and challenging to mitigate. Detection relies on identifying specific sets of features that characterize malicious behavior, and these features have become increasingly complex and diverse as the complexity of malware has increased. Traditional approaches often suffer from high-dimensional feature spaces that increase the computational complexity and reduce the detection accuracy. Therefore, in this paper, a feature optimization approach is proposed that strategically selects the most informative malware features and discards redundant and noisy features to ensure computational efficiency. The ensemble design model using a voting approach is utilized with three base classifiers (LMT, KStar, and Decision Table) that are fed from a feature selection using the Relief algorithm. The proposed models were evaluated through several experiments using three datasets (Derbin, Malgenome, and Prerna) comprising 35,135 samples (10,820 malware samples and 24315 benign samples) across feature settings of 50, 100, 150, and all features. The experimental results highlighted that the detection/classification accuracy can be enhanced via an optimal feature vector. Overall, the model using 150 features was able to achieve the highest performance of 99.61%.

1. INTRODUCTION

The widespread use of Android applications in daily life has become a target for cybercriminals seeking to steal sensitive information and perform malicious attacks. The Kaspersky Mobile Threats Report (2023) revealed an increase in the sophistication and complexity of mobile malware, resulting in more than 33 million cyberattacks worldwide and the discovery of malicious package installations that exceeded 3.5 million. The Android malware can be clustered according to the behavior and characteristics of various families. These families are not equal in size and perform different tasks, thereby reducing the detection efficiency of the developed approaches [1]. In addition, malicious software developers often make minor modifications to evade detection, making the identification of new variants increasingly challenging. On the other hand, traditional approaches to malware detection, including signature-based methods, have increasingly become inadequate for effectively identifying and mitigating these evolving threats. Consequently, researchers have turned their attention to machine learning techniques, particularly those based on feature analysis, as promising avenues for more robust malware detection systems.

Android malware detection methods can be categorized into two methods depending on the feature extraction approach used: static analysis and dynamic analysis. Static analysis involves reverse engineering the Android application package (APK) to extract features from the code and other files without installing or running the application. Dynamic analysis involves running the application to collect information about its behavior and extracting features from it. Although dynamic analysis can accurately detect malicious activities, it requires a significant amount of time and resources to monitor the application in real time. Conversely, static analysis is a more commonly used approach because of its low resource consumption and extensive code coverage [1, 2]. The dynamic method tries to identify and select relevant features from android apps that can effectively discriminate between benign and malicious behavior in these applications. These features

*Corresponding author. Email: hussamjasim@uoanbar.edu.iq

serve as building blocks for machine learning techniques to train and classify malware apps accurately. Feature optimization is a process that involves refining the selected features to enhance the performance of the detection model. This process aims to improve the detection performance through discerning subtle differences between benign and malicious behavior.

As a result, machine learning techniques such as decision trees, random forests and artificial neural networks are used to analyse and classify Android malware as static, dynamic, or mixed [3]. These techniques help in detecting and categorizing malware on the basis of its behavior, code structure, and other features, leading to more accurate and effective malware detection and classification. Therefore, machine learning-based approaches can accomplish better detection for variant unknown and zero-day attacks depending on the collected features (static and dynamic). As a result, investigating and evaluating new approaches is necessary to obtain a model with higher performance. Furthermore, defending against unknown malware requires the discovery of a new solution to overcome the existing limitations of malware detection [3, 4]. Consequently, feature detection and optimization are intertwined processes in the development of effective Android malware detection systems. Researchers can improve the accuracy of developed models by carefully selecting, engineering, and fine-tuning these features. The constructed model is then utilized to detect incoming unknown malware. Usually, the constructed model will undergo retraining after new data are collected, which can help improve its accuracy. This paper investigates feature selection algorithms to evaluate their impact on machine learning classification performance with a focus on ensemble design. The Relief, Information Gain, and Chi-Square algorithms have been explored along with four classification algorithms: Bagging, Voting, KStar, and ENN. Furthermore, to validate the models, several experiments were conducted using three datasets (Derbin, Malgenome, Prerna) with a total of 35,135 APK samples, each of which has 215 features. The main contributions of this paper are to improve Android malware detection by improving the efficiency and accuracy of the classification model.

- Ensemble Model for Malware Classification – This study employs an ensemble learning approach with a voting strategy that combines the benefits of three base classifiers (the logistic model tree, KStar, and decision table) to improve classification performance.
- Feature Selection Optimization - This study investigates the impact of various feature selection algorithms (relief, information gain, and chi-square) on Android malware detection. Using an effective selection approach helps reduce computational complexity while improving accuracy
- Impact of feature selection on accuracy – Determining the number of relative features has an impact on accuracy. Therefore, the experiments were conducted on four different feature sets (50, 100, 150 and all features), which led to the finding that the feature subset (150 features) has the highest classification accuracy, which demonstrates the importance of feature selection.
- Dataset Diversity and Generalization – To ensure the robustness and generalizability of the proposed model across different malware families, the evaluation is carried out via three publicly available datasets (Drebin, Malgenome, Prerna), which have a total of 35,135 APK samples.

The remainder of the paper is structured as follows. Section 2 discusses the previous work that has been conducted on malware detection and machine learning techniques. Section 3 describes the proposed framework of this paper. In section 4, the experimental methodology is presented, and the results are presented in section 5. Finally, section 6 presents an analysis of the contributions and final conclusions.

2. RELATED WORK

In this section, related work on Android malware detection based on machine learning is reviewed. These malware detection systems can be classified according to the method of analysis: static, dynamic, and hybrid analysis. The static analysis method analyses the Android application without executing it. Some researchers have developed state-of-the-art malware detection systems using permissions features [3, 4]. In [4], the authors proposed a system that enables users to safeguard their security and privacy by analysing and removing harmful apps. The analysis of the app is performed by evaluating the permissions requested during application installation via a combination of clustering and classification techniques. However, the drawback of cluster-based approaches is that they cannot detect new families of malware until a new cluster is created. The problem can be mitigated if a larger number of clusters are created to support a wide range of new and unknown malware. Another approach, significant permission identification for Android malware detection (SIGPID), was introduced in [3]. The system prunes the permission data to identify the most significant permissions for distinguishing between benign and malicious apps. Using machine learning-based classification methods, an evaluation reveals that only 22 permissions are significant, and when a support vector machine classifier is used, SIGPID achieves over 90% precision, recall, accuracy, and F-measure while being 4--32 times faster than when all permissions are analysed. These studies consider only permissions that are requested by the application as features. However, they are other features that could help identify an app, such as API calls, strings, components, and intents.

Another Android malware detection approach is based upon API call features [2, 5, 6]. The work in [2] presented a combinational approach. It involves constructing a control flow graph (CFG) of the Android application to obtain API information and using that information to create Boolean, frequency, and time series datasets. Three detection models are then built on the basis of API calls, API frequency, and API sequence aspects, and an ensemble model is created for final determination. The results demonstrate high accuracy and stability, with 98.98% detection precision. However, their experiments were constrained by specific model restrictions, including API calls, API frequency, and API sequences within the source code, which do not encompass all types of malware applications. In [5], the authors utilized information flow analysis to detect mobile malware. Their method evaluates the structure of information flows to recognize patterns of behavior and determine interrelated flows through partial computational path analysis. These flows are referred to by the authors as complex flows. The technique employs N-gram analysis on sequences of API calls along control flow paths. In dynamic analysis, features are acquired from the Android application while it is running in a real or emulated environment. Using API calls alone limits the identification of applications. API calls might be similar for some apps and very different for others; therefore, adding another layer of features to empower the model to identify a pattern is preferable.

The most well-known dynamic features are network traffic, API calls, system calls, and CPU data [7]. Other forms of dynamic analysis can also be performed with other techniques, other than machine learning, such as taint analysis [7] and Dalvik opcode [8]. The effectiveness of machine learning techniques for malware detection has been explored in many papers. Using a multilevel architecture and classifier fusion was proposed in [9]. The author proposed a framework (referred to as DroidFusion). The framework trains several base classifiers, each of which undergoes cross-validation training and testing to measure performance accuracy. A set of ranking-based algorithms are then utilized to generate the final classification model by applying these ranking algorithms to predict accuracy. Their work was evaluated on the DREBIN and Malgenome APK datasets. Similarly, the author in [1] proposed a detection model based on the bagging algorithm and Relief as feature selection, called FB2Droid. Their work involves designing sampling strategies to improve the performance of the bagging algorithm by alleviating imbalance in the dataset. The malware samples gathered from the AMD dataset consisted of 24,553 samples, and the benign samples were gathered from the Google Play Store and APKPure. FB2Droid achieved an accuracy of 97.8%. Like other machine learning methods, deep learning has been investigated in malware detection.

In [10–12], the authors proposed the use of various deep learning models for the purpose of improving detection accuracy. The DLDroid [12] approach is a deep learning model that uses dynamic analysis and stateful input generation to detect malware, which is evaluated on a dataset from the McAfee laboratory with 31,125 samples. They reported that combining dynamics with static features can achieve a higher detection rate than can only dynamic features. The MalDozer [10] framework, which depends on sequence classification via deep learning, can detect malicious apps from using just the API method call. Their evaluation was on 33K samples from three datasets: Malgenome, Derbin, and self-collected samples. Their results revealed an F score between 69% and 99% depending on the number of folds. The DIDriod [11] framework addresses this issue from a different angle. It uses deep learning that is based on images. In DiDroid, the static features are converted into 2D gray images, which are then fed into the deep learning model. For evaluation, they used the Canadian Center for Cyber Security dataset, and they achieved an accuracy of 93.3%. The use of neural networks is particularly advantageous for image processing because of the rich feature sets inherent in images, which are not as prevalent in other applications. This necessitates large training datasets. However, deep neural networks are computationally intensive, leading to slower convergence than other machine learning algorithms do.

The literature review discussed various proposed machine learning-based android malware detection approaches. However, there is still a need to investigate the effectiveness and impact of different feature selection algorithms and feature settings (number of features) on classification performance. Evaluating feature selection and optimization could significantly improve overall performance. In contrast to the existing work on Android malware detection, this paper focuses on investigating the impact and effectiveness of different feature selection algorithms with various numbers of features on the classification performance. A number of studies have claimed to detect both known and unknown types of malware, but their performance on previously unseen malware samples has not been evaluated. An assessment of the ability of any model to generalize novel malware variants is essential for establishing its robustness and practical viability in real-world deployment scenarios.

3. EFFECTIVE FEATURE OPTIMIZATION MODEL (EFOM)

The proposed approach, referred to as the effective feature optimization model (EFOM), aims to increase the accuracy, efficiency, and robustness of detection mechanisms. The EFOM involves a comprehensive approach for feature extraction by using both static and dynamic analysis techniques. The static analysis extracts features and metadata from the application's code, including permissions, API calls, manifest features, and opcode sequences, without executing the application itself. These features provide insight into the structure of the application and intent. Dynamic analysis involves

monitoring the behavior of the application within a controlled environment to capture runtime features such as system calls, network traffic, file system changes, and patterns of memory usage. This dual approach ensures a holistic overview of the application that observes both inherent characteristics and behavioral patterns. Figure 1 presents the general process of the proposed approach.

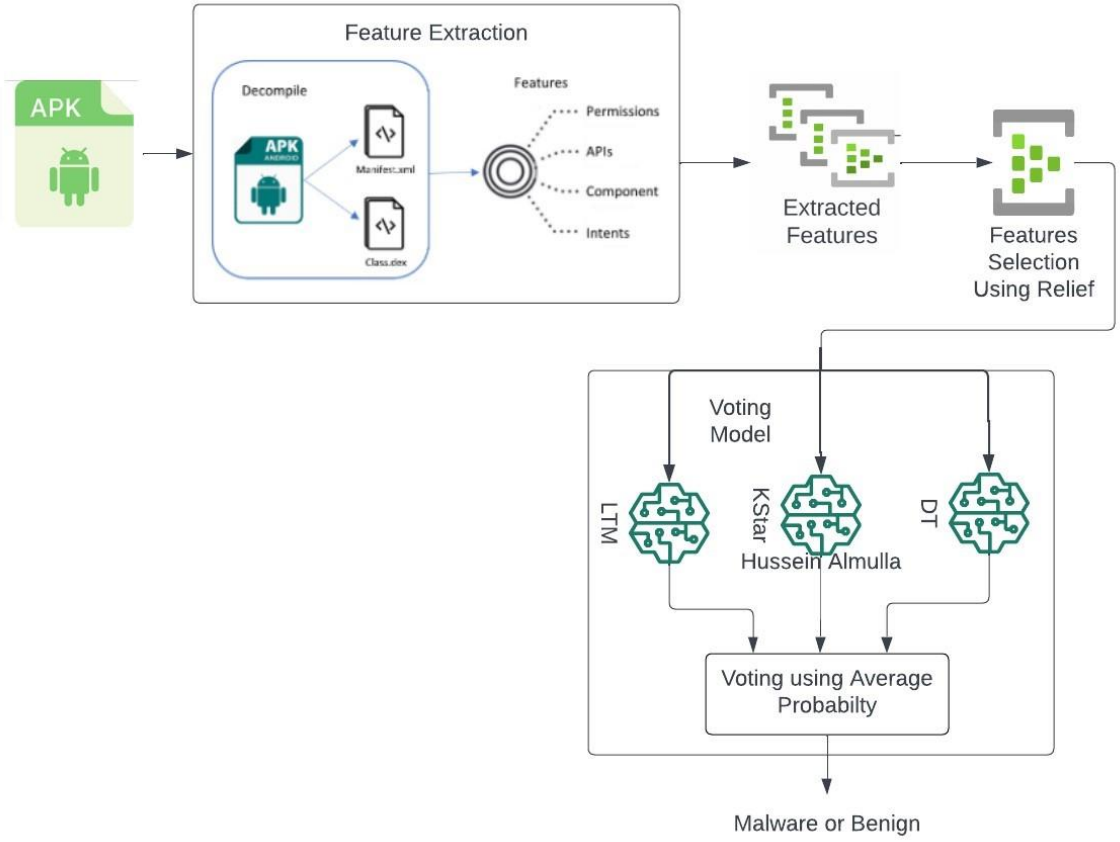

Fig. 1. Overview of the proposed effective feature optimization model.

Generally, the process starts by obtaining Android APKs (benign and malicious), which are then decompiled in a safe environment (sandbox) to avoid spreading the infection of the malware into the machine. Inside the sandbox, the APK is decompiled to obtain AndoidManifest.xml and Class.dex, which contain the permission data, the API calls, and other intents or components. For example, intents are messages that components can use to request functionality from other components, whereas components include activities, services, broadcast receivers, and content providers.

By obtaining various features, the EFOM uses advanced feature selection techniques, such as the Relief algorithm, to identify the most significant features contributing to malware detection. The Relief algorithm assesses the importance of each feature by evaluating how well it distinguishes between instances that are close to each other. This helps in pinpointing the features that have the greatest impact on classification accuracy. Once the most significant features are identified, the optimized feature set is then used to train the detection model. This process ensures that the model is built on the most relevant data, enhancing its ability to classify malware accurately. By focusing on feature optimization, the EFOM not only improves detection performance but also reduces computational complexity by eliminating redundant and irrelevant features. Afterward, this proposed framework utilizes an ensemble scheme for classification that uses multiple classifiers to optimize classification performance. The approach uses the voting algorithm with three base learning models: the logistic model tree (LMT), KStar, and decision table (DT). Since correct classification is highly dependent upon the features of a given instance, the anatomy analysis of APK malware needs to consider the relationships among features. As a result, each of these base models will provide its decision, which then a voting process is conducted using average probability to form the final decision if the APK is malware or benign.

The proposed detection model can be integrated into a cloud-based security system. In such a system, applications are analysed and classified before they are published on an official Android store or distribution platform. Using a detection system ensures the safety of the downloaded application for the end users by detecting any malicious applications early,

which enhances the overall security. Furthermore, utilizing the cloud can empower continuous updates for the detection model, which ensures that the latest threats or malicious applications are included in the model training. The enabled update can mitigate the need for manual updates on user devices.

4. EXPERIMENTAL METHODOLOGY

To explore feature selection and optimization, three methods, chi-square, relief, and information gain, were used to assess performance. The Drebin, Malgenome, and Prerna datasets were utilized to evaluate the proposed approach. The focus of the study was on the impact of reducing the feature vector via different selection algorithms and the impact of an ensemble algorithm on improving the performance. In the experiment, we utilized Python for edited nearest neighbor (ENN) and the Waikato Environment for Knowledge Analysis (WEKA), which is an open-source toolkit that is widely used in machine learning experiments. Several experiments were conducted during the evaluation process utilizing four classification approaches, two of which are ensemble approaches and two are nearest neighbor approaches. Figure 2 shows the experimental methodology of the feature selection process.

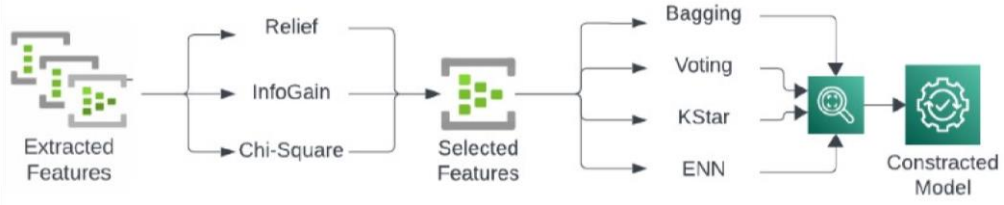

Fig. 2. Overview of the experimental methodology.

4.1. Description of Datasets

To evaluate the proposed models, experiments were performed on three datasets, the details of which are presented in Table I. The first dataset is Drebin-215, which consists of 15036 samples (5560 malware samples and 9476 benign samples). This dataset is part of the Drebin project in 2014 [13]. The second one is Malgenome-215, which consists of 3799 samples (1260 malware samples and 25389 benign samples). This dataset was part of the genome project of 2012 [14]. The third dataset is Prerna, which consists of 16300 samples (400 malware and 12300 benign samples) [15]. In total, there are 35135 samples, among which 10820 are benign samples and 24315 are malware samples. All the above datasets have feature vectors of 215 that belong to four main categories: an API call signature, which is represented by 73 features; a command signature, which is represented by 6 features; an intent, which is represented by 23 features; and a manifest permission, which is represented by 113 features. The utilization of three datasets also helps ensure a good representation of the data to help generalize the results. There are four factors for key differences among the utilized datasets: • Sample size: the Derbin and Prerna datasets have significantly larger sample sizes than the Malgenome dataset does. In addition, Derbin has the highest number of malware samples, making it ideal for training models that need to recognize a wide variety of malware.

- Malware-to-benign ratio: Pre-RAN has a higher ratio of benign samples than malware does, which could influence the training process to be more biased towards benign detection unless properly balanced. Moreover, the Malgenome dataset has a more balanced ratio than the other datasets do, but it still leans more towards benign samples.
- Period of collection: The Malgenome represents older malware samples, providing a historical context. In contrast, Derbin and Prerna included more recent samples, which is crucial for detecting modern malware threats.
- Feature Consistency: All datasets share the same number of features (215), ensuring that the feature space remains consistent across different datasets, which simplifies the process of feature extraction and comparison.

TABLE I. NUMBER OF SAMPLES IN EACH DATASET

Dataset	Ref	Samples	Malware	Benign	Feature
Malgenome	[14]	3799	1260	2539	215
Derbin	[13]	15036	5560	9476	215
Prerna	[15]	16300	4000	12300	215
Total		35135	10820	24135	215

4.2. Feature Extraction Model

The features represent the properties and resources that the APK is provided or needs to operate correctly. The Android malware detection dataset was originally a dataset of APK files that were analysed to extract features. The APK is an Android application installation package. To avoid any malware infection, the APK is implemented in a sandbox environment. The process of extraction requires decompiling the APK to access the main files and the program code, which are later used to obtain the features. Many tools have been used effectively for decomplication, such as Androguard [16], AndroParse [17], and DynaLog [18]. These tools are used to decompile and analyse APK files to obtain AndroidManifest.xml and class files. The tools are capable of extracting common features such as permissions, APIs, strings and intents. The AndroidManifest file provides the required permissions required by the APK for communication and program execution. Five different feature categories were considered during the experiments:

- Permission: the permissions that the app needs to execute an operation or access some resources.
- App components: The building block that is essential for an application. This can include receivers, services, providers, and activities.
- Intent filters: These filters handle the communication among the app components.
- API calls: API calls that are allowed to access data or resources on smartphones.
- Command: the app could need running commands that can perform dangerous operations or access higher privileges than it should have.

4.3. Feature Selection and Optimization Model

Feature selection and ranking are preprocessing steps that aim to reduce the dimensionality of the input data to gain a deeper understanding of the data and to lower the computational cost of the applied model while maintaining or improving model performance. In this step, a subset of relevant and informative features is selected from a large set of features while eliminating features that are redundant, noisy, or have little to no impact on the target variable. This is done by evaluating each feature in predicting the target variable and selecting only the most relevant feature. In addition to cost reduction, overfitting can be reduced by generalizing the learning model. In general, there are two categories of feature selection: feature search and feature subset evaluation. Feature search is used for attribute space search, which can include forward selection, backwards elimination, or both. Feature subset evaluation is used to identify the irrelevant and redundant features. Consequently, this paper uses three different feature selection algorithms that are effectively used in this domain: Relief, Information Gain, and Chi-Square, as shown in Figure 3. The Relief algorithm estimates the quality of attributes. It works by:

1. Select an instance Randomly choose an instance I from the dataset. This is achieved by
2. Find Nearest Neighbors: Identify the two nearest neighbors:
3. Nearest Hit (H): The closest instance to I that belongs to the same class.
4. Nearest Miss (M): The closest instance to I that belongs to a different class.
5. Update Attribute Quality: For each attribute A, update its quality score W[A] on the basis of the values of A in I, H, and M.

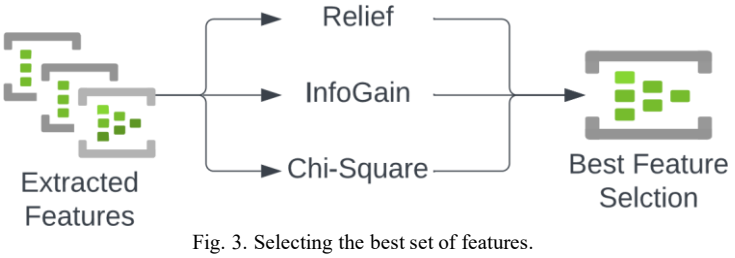

Fig. 3. Selecting the best set of features.

The reason for choosing the Relief algorithm is three points: it is relatively fast (less real-time running), easy to implement (less likely to be misimplemented), and capable of detecting feature dependencies, which is accomplished by using the concept of the nearest neighbor to calculate feature statistics by searching through feature combinations [19]. The information gain (InfoGain) is an entropy-based feature evaluation method. InfoGain calculates the reduction in entropy from the transformation of a dataset. In the context of feature selection, it is used to evaluate the information gain of each variable in the context of the target class. In other words, InfoGain defines the amount of information provided by the feature items for the target class, which can be used for classification [20]. Chi-square feature selection depends on chi-square statistics to help select groups of features by testing the relationships among the given features. The chi-square method evaluates the correlation strength of individual features by computing the statistical value. This involves calculating

the chi-square value between each feature and the target class and observing the relationship between the feature and the target class. The feature can be discarded if the target class is independent of the feature; otherwise, they are dependent, and the feature is important. Subsequently, the desired number of features that exhibit the highest chi-square scores is selected [21, 22].

The main purpose of using different feature selection algorithms is to explore the impact that these algorithms have on the final classification performance. The assumption is that these algorithms behave and process features in different ways. The experiment will encompass each of these algorithms with different configurations using four sizes of features (50, 100, 150, and 215 (all features)) to analyse the impact of reducing the feature dimension. Along with these algorithms, a ranking algorithm was used to rank the features on the basis of their value from the selection evaluation algorithms to eliminate low or nonrelevant features. The top-ranked 50, 100, or 150 terms are then taken and fed into classification algorithms.

4.4. Classification Model

The study used different types of classification algorithms because different machine learning techniques have different strengths and weaknesses, and combining the strengths of multiple approaches can overcome some of their weaknesses. To do that, we can utilize ensemble learning, which is a machine learning technique that works by combining different models into a stronger and more accurate model. It can leverage the diverse properties of combined models, which also help in error mitigation, performance enhancement, and overall robustness. There are different types of ensemble algorithms (see Figure 4). This study considered two algorithms: the bagging and voting algorithms. The Bagging Algorithm is a bootstrap aggregation approach that aims to improve unstable classification or estimation. There are two main steps: bootstrapping, which involves resampling a subdataset with replacement from the initial dataset. This means that a data point in the dataset can be resampled multiple times. These samples are used to train a "weak learner". The next step starts, which is aggregation, during which the "weak learner" is trained via subdatasets. Then, "weak learners" make predictions independently, which are later aggregated to provide an overall prediction [23]. In the voting algorithm, multiple models are used, and they work individually. The final outcome is a result of the voting combination rule. This study uses the average probability, which is calculated for each class, and then the highest probability value is considered the output result. In this paper, we use soft voting, which has a rule of average probability [24]. The hyperparameters for the base models are as follows:

- For LMT, two hyperparameters are used. MinNumInstances” is set to 15, which refers to the minimum number of instances at which a tree node is considered for splitting. NumBoostingIterations is set to -1, which identifies the number of times the process will be performed. Using -1 automatically determines the number of boosting iterations to avoid increasing the computational cost.
- For KStar, two hyperparameters were set. MissingMode determines how the algorithm handles the missing values. In this paper, MissingMode is based on the average entropy of the corresponding feature. To control randomness, GlobalBlend is used with a value of 20 to mix the deterministic and probabilistic approaches.
- For the decision table, three hyperparameters were set. The evaluation measure, which is set to accuracy, defines the metric used to evaluate the predictive power of different subsets of attributes when constructing the decision table. The search method, set to BestFirst, is used to find the best subset of attributes. BestFirst is set with a forward direction and 5 consecutive nonimproving nodes before terminating the search. To evaluate a subset of attributes, corssVal is set to leave-one-out for cross-validation.

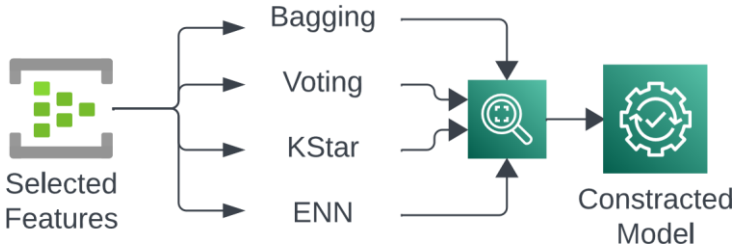

Fig. 4. Classification model based on selected features.

In addition to the ensemble design classification algorithms, the paper uses two nearest neighbor algorithms for the purpose of evaluation. The nearest neighbor algorithms look into finding approximate nearest neighbors on the basis of distance matrices to measure similarity between observations. This paper considers two algorithms, KStar and ENN, for the experiments. K-Star is an instance-based classifier that depends on the K nearest neighbor. It aims to find the shortest path between two vertices. In classification, K-star uses an entropy distance measure depending on the probability of transforming one instance into another to calculate the distance between training data samples. It can provide high

performance and strong generalizability, especially with balanced data [25]. The extended nearest neighbor (ENN) is an advanced version of state-of-the-art KNN classification that considers only the nearest neighbor of a test sample. This could lead to misclassification because the k nearest neighbors of the test samples of the higher-density area could lie in the lower-density area, or vice versa. This is a problem, especially when the dataset is unbalanced or small. In the ENN, the nearest neighbor is extended to consider not only the nearest neighbor but also those samples that consider the test samples as one of their nearest neighbors. However, the ENN is computationally expensive for high-dimensional datasets [26].

5. EXPERIMENTAL RESULTS

The results are divided into two sections: feature selection evaluation, which investigates the performance of each of the three algorithms with different settings; and classification evaluation, which presents the performance of the different classifiers with the use of feature selection.

5.1. Feature selection evaluation

This section investigates the feature selection approach to select and prioritize the features that contain the most discriminative information. This study examined three methods of feature selection, Chi-Square, Relief, and InfoGain, to assess the performance of these methods across multiple datasets and feature settings. As mentioned above, the Drebin, malgenome, and Prerna datasets were used with four feature settings (50, 100, 150, and all features) that were applied to each dataset. Several experiments were conducted using the bagging method as the backbone to measure the accuracy of feature selection approaches.

During the experiment, for the Derbin dataset, the three feature selection techniques returned the same top 50 features but with different ranks, which represent the importance of these features among the others. The ranks of the features between the chi-square coefficient and the information gain are very close. The ranks of the top 14 features all match, and the remaining features are closely ranked. However, Relief ranks the features very differently from the other two methods do. For the Malgenome dataset, Relief returns a set of features that are different from those of other techniques. For the 50-feature selection, 16 out of the 50 features belong to the selected features in the Relief list but not in the information gain and chi-square. With respect to Prerna, Relief returned 28 out of 50 features that are different from those from which Chi-Square and Information Gain returned. Table II shows the top 20 ranked features based on the Relief algorithm for the three datasets. We chose Relief because it has different ranking features than the other algorithms do.

TABLE II. DATASETS WITH THE TOP 20 FEATURES

Rank	Durbin Dataset	Malgenome Dataset	Prerna Dataset
1	TelephonyManager.getLine1Number	READ_SMS	intent.action.BOOT_COMPLETE
2	GET_ACCOUNTS	TelephonyManager.getSubscriberId	SEND_SMS
3	System.loadLibrary	ClassLoader	RECEIVE_BOOT_COMPLETED
4	intent.action.BOOT_COMPLETED	content.Context.registerReceiver	READ_SMS
5	Ljava.lang.Class.cast	content.Context.unregisterReceive	READ_PHONE_STATE
6	TelephonyManager.getSubscriberId	android.os.Binder	ACCESS_WIFI_STATE
8	READ_PHONE_STATE	Ljava.lang.Class.getCanonicalName	android.telephony.SmsManager
9	Ljava.lang.Class.getMethods	intent.action.BOOT_COMPLETED	System.loadLibrary
10	DexClassLoader	Transact	WRITE_SMS
11	WRITE_HISTORY_BOOKMARKS	ServiceConnection	WRITE_EXTERNAL_STORAGE
12	android.telephony.SmsManager	bindService	READ_CONTACTS
13	Ljava.lang.Class.getCanonicalName	onServiceConnected	WAKE_LOCK
14	content.Context.registerReceiver	TelephonyManager.getLine1Number	CHANGE_WIFI_STATE
15	Runtime.getRuntime	SEND_SMS	android.content.pm.PackageInfo
16	content.Context.unregisterReceive	WRITE_SMS	abortBroadcast
17	READ_SMS	Chmod	Mount
18	getBinder	TelephonyManager.getDeviceId	ClassLoader
19	Ljava.lang.Class.getField	attachInterface	VIBRATE
20	TelephonyManager.getLine1Number	Runtime.exec	/system/bin

Table III and Figure 5 illustrate the results of the feature selection model performance obtained from the experiments. The results showed that the accuracy of the three methods on the Drebin dataset, which are based on feature selection and the optimization model, is generally high, with values ranging from 96.27% to 98.64%. In addition, the best accuracy of 89.64% was achieved when 100 and 150 features were set via the Relief method. In contrast, in this dataset, InfoGain tends to have slightly lower accuracy than Chi-Square and Relief. The bagging with chi-square and relief feature selection methods appears to be a strong combination for this dataset. Among all feature settings, the 100 and 150 settings presented

consistent accuracies, with accuracies higher than 96%, indicating that feature selection approaches can help improve model performance. As a result, bagging with chi-square and relief feature selection methods appears to be a strong combination for this dataset. On the other hand, the accuracy of the utilized feature selection approaches on the Malgenome dataset is relatively high, ranging from 96.27% to 99.34%. When the 50-feature setting was used, the accuracy of the chi-square method reached 96.37%. This finding indicates that chi-square is effective in selecting a subset of features that significantly contribute to the classification task. The consistency of the results across different feature counts highlights its reliability. In contrast, the Relief method performs slightly worse than the chi-square method, with a value of 96.27%. However, the difference is marginal, and Relief demonstrates its ability to select features even in a reduced feature space. Its localized feature importance assessment might contribute to its effectiveness in specific contexts. Similarly, InfoGain achieved an accuracy of 96.27%, similar to the chi-square method. While not the highest, its consistent performance across various feature settings suggests its stability and reliability. The ability of InfoGain to measure the information gain makes it a robust choice in capturing relevant features.

TABLE III. PERFORMANCE OF THE FEATURE SELECTION MODEL

Feature settings	Feature Selection	Drebin	Malgenome	Prerna
50 Features	Chi-Square	96.37%	96.37%	88.58%
	Relief	96.27%	96.27%	88.32%
	InfoGain	96.27%	96.27%	88.64%
100 Features	Chi-Square	98.37%	98.95%	90.58%
	Relief	98.64%	99.34%	91.32%
	InfoGain	98.00%	98.95%	90.64%
150 Features	Chi-Square	98.40%	99.21%	91.53%
	Relief	98.64%	99.08%	91.63%
	InfoGain	98.50%	98.95%	91.60%
All Features		97.44%	97.81%	90.50%

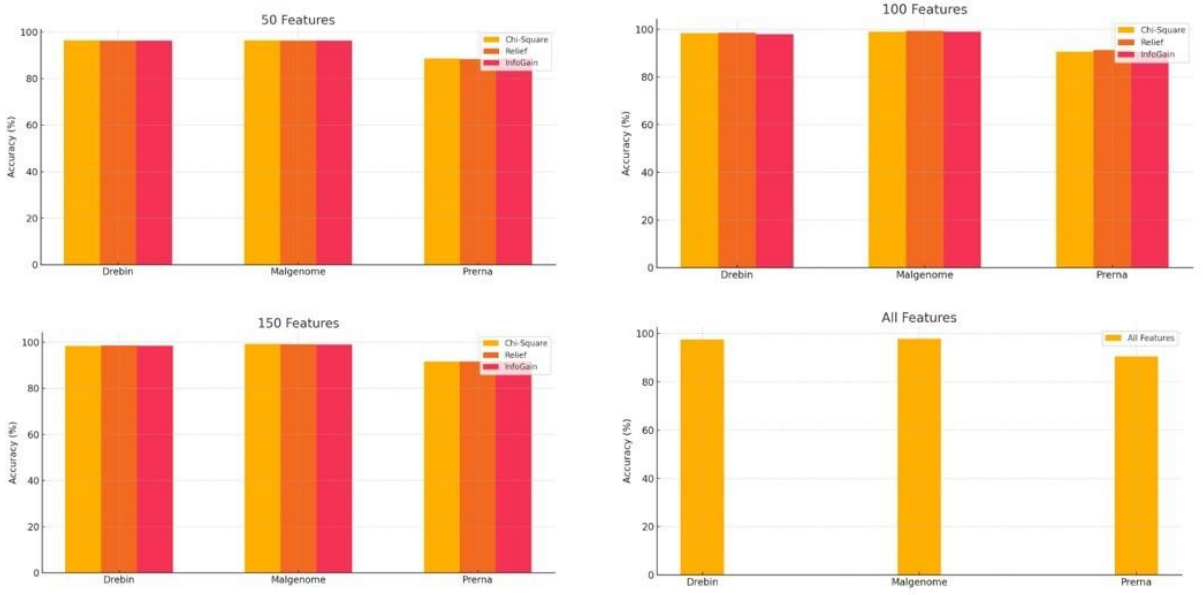

Fig. 5. Feature Selection Performance.

The proposed approach achieved significant accuracy when 100 features were set via the Malgenome dataset across the utilized feature selection methods. Furthermore, chi-square and InfoGain increase accuracy to 98.95%. This suggests that, with a larger feature set, they are able to capture additional relevant information, leading to improved classification accuracy. However, the Relief method outperforms both Chi-Square and InfoGain, achieving an impressive accuracy of 99.34%. This suggests that the InfoGain method has the capacity to leverage additional features effectively. The 150-feature setting improves the accuracy of the proposed approach. In other words, the chi-square method continues to improve, achieving an accuracy of 99.21%, in which the InfoGain method gains the exact accuracy of the chi-square method. This suggests that the additional features contribute positively to the model's ability to discern patterns in the Malgenome dataset. However, InfoGain was robust in consistently selecting relevant features across varying feature

counts. However, the proposed approach offered low accuracy when all features of the Malgenome dataset were used. This means that several ineffective features lead to unsuccessful classification.
The results on the Prerna dataset were less accurate than those on Drebin, and the percentage of malgenomes ranged from 88.32% to 91.63%. The feature setting of 50 yielded the lowest accuracy among the other settings across the three methods. The accuracy of the classification results improves as the number of selected features increases. However, among the three feature selection methods, Relief consistently performs the best, followed closely by InfoGain and Chi-Square. These results suggest that a feature set of 150 provides the highest accuracy, with Relief being the most effective feature selection method for the Prerna dataset.
The results indicate that as the number of features increases, the classification accuracy generally improves for all three feature selection methods. Relief consistently outperforms chi-square and InfoGain in terms of accuracy, especially in the 100-feature setting. The chi-square test shows a remarkable improvement in accuracy as the feature count increases. The results suggest that the effectiveness of feature selection methods varies depending on the dataset and the number of features considered.

5.2. Classification evaluation

This section presents the results of the classification methods. On the basis of the above feature selection experiments, 100 and 150 feature settings were used to investigate the malware classification ability. Tables IV, V, and VI present the classification model performance results obtained from the experiments.
By using the Drebin dataset, the accuracy across the four classification methods was slightly sensitive to the number of features. However, using the 100-feature setting and chi-square with bagging, voting and KStar demonstrated a consistent performance ranging between 98.14% and 98.67%, whereas the ENN algorithm presented a lower accuracy of 97.93%. Moreover, the Relief classification algorithm, which uses the same Chi-Square setting, achieved a steady accuracy of more than 98.64%. In addition, InfoGain offered acceptable accuracy of more than 98%. Generally, the Voting and ENN algorithms yield notable improvements in accuracy when they transition from 100 to 150 feature settings, suggesting that a richer feature set enhances their classification capabilities. KStar showed a slight decline with 150 features. In other words, the slight decrease in accuracy from 98.67% to 98.50% with 150 features might indicate a potential optimal feature set of approximately 100 features for KStar in this context. In contrast, the substantial improvement from 97.93% to 98.60% with 150 features indicates that the ENN benefits significantly from a larger feature set. Therefore, the experiments demonstrated the nuanced relationship between the number of features and the performance of classification algorithms for Android malware detection via the Drebin dataset.

TABLE IV. CLASSIFICATION MODEL PERFORMANCE METRIC RESULTS FOR THE DREBIN DATASET.

Metrics	Feature Selection	100 Features				150 Features			
		Bagging	Voting	Kstar	ENN	Bagging	Voting	Kstar	ENN
Accuracy	ChiSquare	0.984	0.981	0.987	0.979	0.984	0.984	0.985	0.986
	Relief	0.986	0.986	0.988	0.987	0.986	0.985	0.987	0.987
	InfoGain	0.980	0.982	0.983	0.981	0.985	0.983	0.985	0.985
Precision	ChiSquare	0.989	0.991	0.989	0.9917	0.992	0.992	0.991	0.988
	Relief	0.993	0.989	0.989	0.992	0.991	0.995	0.992	0.990
	InfoGain	0.989	0.989	0.989	0.973	0.989	0.995	0.989	0.984
Recall	ChiSquare	0.989	0.989	0.989	0.984	0.99	0.996	0.991	0.988
	Relief	0.993	0.989	0.989	0.984	0.991	0.995	0.992	0.972
	InfoGain	0.989	0.989	0.989	0.984	0.989	0.995	0.989	0.980
F-Measure	ChiSquare	0.989	0.989	0.989	0.988	0.992	0.996	0.991	0.988
	Relief	0.993	0.989	0.989	0.988	0.991	0.995	0.992	0.986
	InfoGain	0.989	0.989	0.989	0.979	0.989	0.995	0.989	0.982

TABLE V. CLASSIFICATION MODEL PERFORMANCE METRICS RESULTS FOR MALGENOME DATASET.

Metrics	Feature Selection	100 Features				150 Features			
		Bagging	Voting	Kstar	ENN	Bagging	Voting	Kstar	ENN
Accuracy	ChiSquare	0.989	0.991	0.989	0.992	0.992	0.996	0.991	0.992
	Relief	0.993	0.989	0.992	0.992	0.991	0.995	0.992	0.990
	InfoGain	0.989	0.995	0.989	0.986	0.989	0.995	0.989	0.988
Precision	ChiSquare	0.984	0.981	0.987	0.9781	0.984	0.984	0.985	0.979
	Relief	0.986	0.988	0.988	0.972	0.986	0.985	0.987	0.991
	InfoGain	0.98	0.982	0.983	0.972	0.983	0.985	0.985	0.986
Recall	ChiSquare	0.984	0.981	0.987	0.966	0.984	0.984	0.985	0.983
	Relief	0.986	0.988	0.988	0.974	0.986	0.985	0.987	0.975
	InfoGain	0.98	0.982	0.983	0.976	0.985	0.983	0.985	0.973
F-Measure	ChiSquare	0.984	0.981	0.987	0.972	0.984	0.984	0.985	0.981
	Relief	0.986	0.988	0.988	0.974	0.986	0.985	0.987	0.982
	InfoGain	0.98	0.982	0.983	0.974	0.983	0.985	0.985	0.981

The results on the Malgenome dataset were better than those on the Drebin dataset. The methods KStar and Voting are proving to be strong contenders when 100-feature settings are used. The sequential increase in accuracy from 98.95% to 99.08% shows that combining multiple classifiers results in the best performance in malware detection. Voting, although slightly conservative, maintains a solid accuracy of 98.95% across the three feature selection methods. ENN, with an accuracy of 99.21%, highlighted the power of focusing on quality over quantity. Similarly, the chi-square and relief feature selection methods, which use 150 settings across the four utilized classification algorithms, offered a reliable accuracy of more than 99%. Moreover, the Voting algorithm outperformed the other algorithms, with an accuracy of 99.61% when 150 settings were used. It focuses on eliminating noise and enhancing the quality of the dataset for classification. While Bagging and KStar are performing admirably. Voting, in particular, seems to be excelling, with a slight edge over Bagging. These results highlight the strengths of ensemble methods (bagging and voting) in capturing diverse patterns within the Malgenome dataset. Although KStar is slightly more conservative, it remains effective in its instance-based learning approach. As a result, the ENN emerges as a standout performer, emphasizing the importance of dataset refinement in achieving high accuracy.

TABLE VI. CLASSIFICATION MODEL PERFORMANCE METRIC RESULTS FOR THE PRERNA DATASET.

Metrics	Feature Selection	100 Features				150 Features			
		Bagging	Voting	Kstar	ENN	Bagging	Voting	Kstar	ENN
Accuracy	ChiSquare	0.906	0.906	0.908	0.903	0.915	0.918	0.917	0.907
	Relief	91.32	0.909	0.916	0.897	0.916	0.916	0.918	0.901
	InfoGain	90.64	0.906	0.908	0.892	0.916	0.921	0.917	0.906
Precision	ChiSquare	0.906	0.907	0.908	0.957	0.918	0.918	0.918	0.955
	Relief	0.913	0.910	0.916	0.951	0.916	0.916	0.917	0.946
	InfoGain	0.906	0.907	0.908	0.958	0.916	0.921	0.917	0.949
Recall	ChiSquare	0.906	0.9132	0.915	0.918	0.918	0.917	0.918	0.918
	Relief	0.913	0.910	0.916	0.911	0.916	0.916	0.917	0.936
	InfoGain	0.906	0.908	0.908	0.897	0.916	0.921	0.917	0.936
F-Measure	ChiSquare	0.906	0.926	0.915	0.918	0.918	0.916	0.9365	0.937
	Relief	0.916	0.930	0.916	0.920	0.916	0.917	0.9365	0.937
	InfoGain	0.908	0.926	0.916	0.921	0.916	0.917	0.9365	0.937

With respect to the Prerna dataset, the experiments revealed that the accuracy of all the classification algorithms using the three feature selection methods was lower than that of the other datasets. In other words, the results presented a moderate increase in accuracy by moving from 100 to 150 features across all algorithms, with an accuracy of 91%, highlighting the importance of feature richness in enhancing classification performance. Compared with the other algorithms, the ENN algorithm achieves slightly lower performance.

In general, the results of the experiments highlighted the nuanced trade-offs between feature quantity and algorithmic performance, emphasizing the need for a balanced approach in malware classification. These results paint a picture of a well-coordinated team of classifiers with feature selection methods working together to successfully identify malware in the Drebin and Malgenome datasets. In addition, the sequential accuracy values show a progression in performance, highlighting the effectiveness of these methods in tackling the complexities of malware detection to gain deeper insights into the characteristics of Android malware.

6. COMPARATIVE ANALYSIS

The overall best results were obtained by using the Voting algorithm with Relief feature selection, with the highest accuracy of 99.47% in Malgenome and 92.12% in Prerna, both with 150 features. Table VII presents a comparison of results from previous studies versus this study. DROIDFUSION [9] was able to obtain an F-measure of 98.4% for the Malgenome dataset, which was higher than those of the other nine algorithms that they investigated. Compared with this study, for the same dataset, Voting with Relief and the setting of 100 features (Voting_Relief_100) yield an F-measure of 99.5%. For the Derbin dataset, DROIDFUSION obtains an F-measure of 98.72%, which was the highest obtained value in their experiment. However, the combination of Voting_Relief_100 provides an F-measure of 98.8%, which is slightly higher. In a different comparative analysis and utilization of various machine learning algorithms, [27, 28] and [35] reported that using a multilayer perceptron (MLP) with a 66% split on the Malgenome and Derbin datasets obtained accuracies of 99.4% and 0.97%, respectively. These results for Malgenome are close to our finding with very small differences; however, there is noticeable performance improvement with Derbin, where the accuracy gain is approximately 0.13% greater. The random forest method is an alternative that can achieve good accuracy between 98.8 and 99.3 on the Malgenome.

TABLE VII. COMPARISON OF F-MEASURE VALUES FOR DIFFERENT APPROACHES

Approach	Malgenome	Derbin
EFOM (this study)	99.5	98.9
DROIDFUSION [9]	98.4	98.72
Multilayer Perceptron [27]	98.9	97
Random Forest [27, 28]	98.8 - 99.3	-
KNN & RF [29]	95.9	-
MalDozer [10]	99.84	99.21
EAODroid [30]	-	99.5
LSTM [29]	99.3	98.2

A neural network (deep learning and LTSM) is used by MalDozer, EAODroid [10, 30]. MalDozer was evaluated on the Malgenome and Derbin datasets. MalDozer was able to provide F-measures of 99.84% and 99.21 for Malgenome and Derbin, respectively. MalDozer has a slightly higher F-measure than does the proposed approach of this study. In the same context, the EAODroid tool focuses on the API-based detection technique, which considers only the invoked statements and leaves behind the permission, string, intent, etc. EAODroid was evaluated on sample malware collected from Derbin (just the malware samples) and AMD, and the benign samples were collected from Xiaomi and PlayDrone. EAODroid with a feature dimension of (64 × 64) obtained an accuracy of 99.5%, but when the dimension was reduced to (11 × 11), the accuracy was 98.6. Although EAODroid was evaluated on the same dataset (part of Derbin), comparing the results can provide a context for future research. In addition, there is a need to investigate the setting of features with different algorithms. LTSM [29] was also successful in detecting malware in Malgenome and Drebin, with an accuracy of 99. 3% and 98.2%, respectively

Notably, while our proposed approach outperforms many of the current state-of-the-art methods, it is still marginally behind the deep learning approaches. The better performance of deep learning can be attributed to its ability to extract better features. However, using a deep neural network comes with a significant computational cost and typically requires large training data, which can lead to slower convergence than other machine learning algorithms.

7. CONCLUSION

This paper presented an approach to enhance Android malware detection through an effective feature optimization model (EFOM). It addresses the critical challenge of identifying and optimizing relevant features for robust malware detection by considering the dynamic and diverse nature of Android malware. Feature selection plays a crucial role in identifying the most relevant attributes that can contribute to the differentiation between malware and benign applications. By utilizing approaches such as the Relief algorithm, it effectively reduces the feature space, thereby eliminating redundant and irrelevant features. This not only improves the computational efficiency but also enhances the overall accuracy of the classification models.

Generally, a combined approach of feature optimization and classification leads to more accurate, efficient, and reliable detection of malware applications. This dual optimization strategy enhances the ability to detect malicious applications promptly and accurately, thereby contributing to improved security in the Android ecosystem. Future research should prioritize the optimization of features used in malware detection. This involves exploring novel feature engineering techniques to increase the quality and relevance of features. Investigating advanced ensemble methods for integrating

multiple detection models can further refine feature selection. Additionally, leveraging deep learning approaches for detailed feature analysis can uncover complex patterns and relationships within the data, potentially leading to improved detection accuracy. By focusing on feature optimization, we can develop more robust and effective malware detection systems.

The features are the cornerstone on which classification depends. Therefore, future work should investigate the impact of using optimization approaches such as feature selection instead of a filter-based approach. The optimization approach can provide a dynamic mechanism for selecting the most relevant features to increase detection accuracy.

References

[1] K. Shao, Q. Xiong, and Z. Cai, "Fb2droid: A novel malware family-based bagging algorithm for android malware detection," *Security and Communication Networks*, vol. 2021, no. 1, p. 6642252, 2021.

[2] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE access*, vol. 7, pp. 21235–21245, 2019.

[3] 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.

[4] S. B. Almin and M. Chatterjee, "A novel approach to detect android malware," *Procedia Computer Science*, vol. 45, pp. 407–417, 2015.

[5] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1231–1245, 2018.

[6] 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. 7, pp. 69246–69256, 2019.

[7] V. G. Shankar, G. Somani, M. S. Gaur, V. Laxmi, and M. Conti, "Androtaint: An efficient android malware detection framework using dynamic taint analysis," *2017 ISEA Asia security and privacy (ISEASP)*, pp. 1–13, 2017.

[8] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018.

[9] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.

[10] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018.

[11] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, "Didroid: Android malware classification and characterization using deep image learning," in *Proceedings of the 2020 10th International Conference on Communication and Network Security*, 2020, pp. 70–82.

[12] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dl-droid: Deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, p. 101663, 2020.

[13] 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.

[14] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 95–109.

[15] P. Agrawal and B. Trivedi, "Evaluating machine learning classifiers to detect android malware," in *2020 IEEE International Conference for Innovation in Technology (INOCON)*. IEEE, 2020, pp. 1–6.

[16] A. Desnos, "Androguard, a full python tool to play with android files," 2023.

[17] R. Schmicker, F. Breitinger, and I. Baggili, "Androparse-an android feature extraction framework and dataset," in *Digital Forensics and Cyber Crime: 10th International EAI Conference, ICDF2C 2018, New Orleans, LA, USA, September 10–12, 2018, Proceedings 10*. Springer, 2019, pp. 66–88.

[18] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "Dynalog: An automated dynamic analysis framework for characterizing android applications," in *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security)*. IEEE, 2016, pp. 1–8.

[19] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of biomedical informatics*, vol. 85, pp. 189–203, 2018.

[20] S. Jadhav, H. He, and K. Jenkins, "Information gain directed genetic algorithm wrapper feature selection for credit rating," *Applied Soft Computing*, vol. 69, pp. 541–553, 2018.

[21] X. Jin, A. Xu, R. Bie, and P. Guo, "Machine learning techniques and chi-square feature selection for cancer classification using sage gene expression profiles," in *Data Mining for Biomedical Applications: PAKDD 2006 Workshop, BioDM 2006, Singapore, April 9, 2006. Proceedings*. Springer, 2006, pp. 106–115.

[22] M. R. Mahmood, "Two feature selection methods comparison chi-square and relief-f for facial expression recognition," in *Journal of Physics: Conference Series*, vol. 1804, no. 1. IOP Publishing, 2021, p. 012056.

[23] P. Bühlmann, "Bagging, boosting and ensemble methods," *Handbook of computational statistics: Concepts and methods*, pp. 985–1022, 2012.

[24] I. Gandhi and M. Pandey, "Hybrid ensemble of classifiers using voting," in *2015 international conference on green computing and Internet of Things (ICGCIoT)*. IEEE, 2015, pp. 399–404.

[25] B. Ghasemkhani, O. Aktas, and D. Birant, "Balanced k-star: an explainable machine learning method for internet-of-things-enabled predictive maintenance in manufacturing. machines 11 (3), 322," 2023.

[26] B. Tang and H. He, "Enn: Extended nearest neighbor method for pattern recognition [research frontier]," *IEEE Computational intelligence magazine*, vol. 10, no. 3, pp. 52–60, 2015.

[27] I. M. Olorunshola Oluwaseyi Ezekiel, Oluyomi Ayanfeoluwa Oluwasola, "An Evaluation of some Machine Learning Algorithms for the detection of Android Applications Malware," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 6, pp. 1741–1749, 2022.

[28] J. Y. Ndagi and J. K. Alhassan, "Machine learning classification algorithms for adware in android devices: a comparative evaluation and analysis," in *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*. IEEE, 2019, pp. 1–6.

[29] M. Dhalaria and E. Gandotra, "A framework for detection of android malware using static features," in *2020 IEEE 17th India Council International Conference (INDICON)*. IEEE, 2020, pp. 1–7.

[30] L. Huang, J. Xue, Y. Wang, D. Qu, J. Chen, N. Zhang, and L. Zhang, "Eaodroid: Android malware detection based on enhanced api order," *Chinese Journal of Electronics*, vol. 32, no. 5, pp. 1169–1178, 2023.