Universiteit Leiden

Master Computer Science

Explanation of XAI clustering methods on Android malware family categorization

Name:	Feiyang Sun
Student ID:	s3134644
Date:	19/07/2023
Specialisation:	Data Science
1st supervisor:	Olga Gadyatskaya
2nd supervisor:	Yury Zhauniarovich
Daily Supervisor:	Rui Li

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this work, we study the combination of explainable artificial intelligence (XAI) and Android malware family clustering. XAI is a popular domain in artificial intelligence that aims to provide logical reasoning behind traditional AI models and generate meaningful explanations. Though there have been numerous researches on the XAI classification of Android malware families, they ignore the large emergence of unlabeled data nowadays. Additionally, considering the fact of gap in studying clustering methods that provide explanations for Android malware families, we present this paper, which compares three algorithms in this domain (`ExKMC`, `ICOT` and `CART`).

First, we analyze the dataset AMD through data exploration and data preprocessing. Next, we conduct multiple experiments and compare the results obtained from `ExKMC` with those of a baseline method `CART` and the newest approach `ICOT`. We then draw conclusions about the key features of malware that the algorithm identifies and compare them with the ground truth using multiple evaluation metrics. Algorithm `ICOT` and `ExKMC` perform well compared to the baseline model `CART` and can provide reasonable features to cluster the Android malware family.

Keywords: XAI, Clustering Methods, Android Malware Family, Interpretation

Acknowledgements

I would like to express my deep gratitude to my thesis supervisors, Olga, Yury and Rui. Thank you for providing me with endless support, whether it was through your kind and positive words about my work or your unwavering belief in me. I am also grateful for olga's willingness to serve as my reference when I search for job opportunities. Rui, her patient guidance, support, and brilliant ideas have been invaluable to me.

Also send deep gratitude to my mom and dad, for taking everything unconditionally, and also thanks for their financial support for finishing the master;). I am also thankful to my friends and classmates for presenting me with challenges that helped me grow. Their companionship during this half-year journey of research means a great deal to me. Additionally, I appreciate my friends who accompanied me on adventures and patiently listened to my doubts and joys.

Although I don't have a cat of my own, I can imagine the comfort it would provide. Let's indulge in this imaginative cat and appreciate the solace it brings. Furthermore, I am grateful for my laptop, which allowed me to conduct numerous experiments without complaints, even when it became sluggish. It has been a reliable companion throughout this process.

The presence of the Leiden University library has been instrumental in supporting my studies. Its serene environment and ambiance have enabled me to concentrate on my research continuously. The library's beautiful building, steeped in centuries of history, reminds me to be grateful for the privilege of learning within its walls. I am also grateful for the nearby river, which refreshes my mind. My bike, carrying me effortlessly, has been a source of relaxation after long days of studying. The gentle breeze accompanying my rides has provided a sense of freedom that I never experienced in my hometown. I am grateful to Leiden University for granting me the opportunity to conduct research and for cultivating an environment that fosters personal growth.

I cannot help but reflect on how unexpected it was to work on a kmeans-based algorithm, which happens to be the topic of my bachelor's thesis. As I delved deeper into my research, I came across the datasets sourced from a research project conducted at an American university. The realization that these datasets originated from a place I never imagined seeing it in my lifetime truly left me amazed. it truly amazed me. Such coincidences have filled my life, and I believe they add a touch of wonder to this scientific thesis I am writing.

In summary, I extend my thanks to everyone who has contributed to my achievements thus far. I appreciate all the support that will aid my future mental and academic development.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays, Android has become a dominant operating system in the realm of smartphones. It owns an impressive user base of 3.6 billion across 190 countries [110]. By the end of 2022, it held a worldwide market share of 71.25%. This rapid growth has witnessed the explosive potential and power of Android on a global scale. For this reason, it has attracted attention not only from users but also from attackers to hack smartphones for extracting benefits. Since 2012, the emergence of Android malware has experienced exponential growth, reaching 1,047 million cases by June 2023 (equivalent to 259,591 per day) [102]. The proliferation of Android malware poses significant concerns for smartphone users, Android developers, and the cybersecurity community.

A large amount of emergence of Android malware has had a substantial negative impact in various aspects. Similar to scammers in real life, Android malware is capable of stealing classified information from your personal device while disguising itself as legitimate software. For instance, malware Trojan horse [46] performs functions like stealing classified information or gaining unauthorized access to smartphones while disguising itself. It can capture user keystrokes and transmit the gathered information to remote servers via the Internet or SMS server. Another notable Android malware is ransomware [101], which aims to lock the device and encrypt classified data. Ransomware requires a ransom payment from the victim in exchange for regaining access to the user's device and its information. Additionally, Android malware may display advertisements on the user's device, containing links or directives that prompt users to click on them. Moreover, it can cause damage to the device itself, such as increased battery drainage, overall device slowdown, or unauthorized modifications to the system.

Considering the increasing influx of Android malware and the resulting damages, researchers and companies have devised strategies to study and mitigate their impact by

categorizing them into families. Additionally, attackers usually modify existing families by appending new behaviors [21]. Therefore, a comprehensive study of Android malware families enables us to gain more understanding of Android malware evolution and analysis. Based on the analysis, companies can develop more robust strategies to prevent existing and future damage from similar Android malware.

Hence, in order to address these challenges from Android malware, researchers have made substantial efforts. The recent emergence of machine learning (ML) presents a huge potential for learning patterns and identifying features. Especially compared to traditional signature-based methods, ML could distinguish hidden relationships behind data through technologies of classification and clustering. Consequently, this domain has garnered significant attention from researchers. They have proposed a variety of machine learning and deep learning techniques to effectively classify Android malware [125, 93, 60, 98, 76, 28, 44].

Among these machine learning technologies, the classification models have made great contributions to analyzing and grouping Android malware families. However, classification technology can only be used for labeled samples, with a clear definition of whether it's malware or not or the type of malware family. And the prerequisite for the processed data is sufficient labeled data for automatic ML, which is not enough. The reason lies in the new coming of 2,722,824 malware samples per week [102] that need to be categorized and analyzed. Thus, there comes a necessity to automatically cluster the newborn malware into its own families or into the unknown new Android malware family [122].

The instinct solution for analyzing the family of unlabeled samples is to utilize the similarity of newly emerged malware samples to compare and assign labels based on known families. On the other hand, by categorizing the newly emerged samples belonging to known families, new samples can also help to contribute the newly own attributes from themselves to the pool of existing samples and enhance the discriminative ability to identify incoming samples. Consequently, due to the considerable appeal of automated labeling technology, clustering methods have emerged as a prevailing trend [13, 118, 86, 57, 7].

Currently, machine learning techniques have demonstrated remarkable proficiency in clustering Android malware families. However, ML's limited interpretability has introduced a potential vulnerability, where subtle modifications of malware may be able to evade detection. For example, attackers may discover techniques or engage in activities to evade identification by absorbing AI-based frameworks, a phenomenon commonly referred to as evasion attacks [105]. This susceptibility is further compounded by the prevalence of similar activities such as adversarial machine learning attacks, encompassing evasion, poisoning, and backdoor incursions [67].

Additionally, researchers often prioritize the accuracy of models over interpretability [92]. The lack of crucial internal model logic hampers the display, examination, and analysis

of machine learning results [22]. Therefore, to enhance comprehension of the underlying AI model and facilitate understanding of its decision-making process, models should be transparent and interpretable. The solution for explainable methods in artificial intelligence is commonly referred to as "XAI," which has gained widespread usage across multiple domains [130].

Presently, there are numerous researchers who have focused on clustering malware families and conducting analysis to understand their causes and crucial features. Additionally, a considerable amount of work has been carried out in the field of XAI for malware family classification. However, there is a significant scarcity of researchers who specifically delve into the application of XAI methods for real-world clustering malware families datasets, particularly in addressing analyzing, and providing explanatory insights when clustering unlabeled datasets of Android malware families.

1.2 Contribution

Therefore, motivated by the existing gap in research on clustering methods for XAI in the domain of Android malware analysis, we conducted this study by applying multiple clustering XAI techniques to real-world datasets of Android malware families, The framework of whole thesis is followed as Figure 1.1.

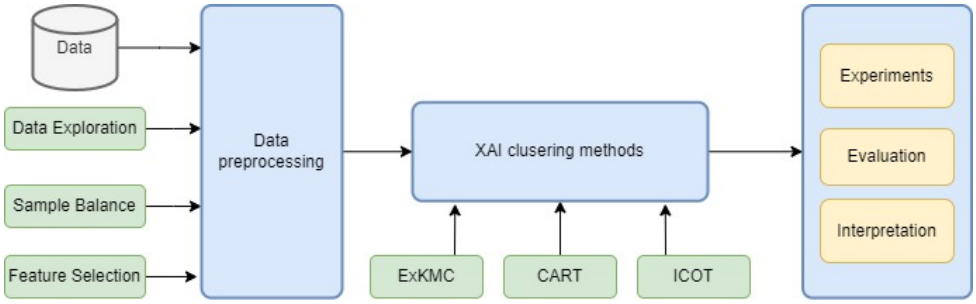


Figure 1.1: Framework of this thesis. It begins with data preprocessing and implements three selected algorithms. Subsequently, the data is fed into the algorithms to conduct experiments. The results of the algorithms are then evaluated, and a comprehensive explanation of the results is provided.

In this study, we utilized the dataset AMD provided by Wei et al. [116] for our research purposes. The dataset comprises 24,650 samples of malware applications, categorized into 71 distinct malware families. Additionally, this dataset includes comprehensive descriptions of malware behaviors, enabling us to compare the outcomes generated by our models with descriptions provided in the dataset.

In our study, we employed three XAI clustering algorithms: Expanding k-Means clustering (`ExKMC`) [38], Clustering via Optimal Trees (`ICOT`) [15], and Classification And

Regression Tree (CART) [55]. The selection of ExKMC was based on its state-of-the-art clustering performance, efficient computational cost, and promising outcomes obtained from experiments conducted with synthetic and real-world datasets. We employed CART as the baseline algorithm to assess the advancements introduced by the newly proposed methods. Furthermore, we compared ExKMC with ICOT due to algorithm ICOT's novelty and flexibility in achieving high-quality partitions of the feature space.
These clustering algorithms not only facilitate the exploration of poorly understood data [11] but also address the increasing prevalence of unlabeled data in current scenarios. The primary contributions of this research are outlined as follows:

- We implemented and compared three clustering XAI models on Android malware family clustering on the AMD dataset.
- A detailed evaluation of the algorithm's results from the perspectives of quality and interpretability.

1.3 Outline

The thesis is structured as follows. In the next chapter, we conduct a review of recent articles concerning Android malware detection and the application of XAI methods to Android malware. Chapter 3 introduces mathematical notations, and preliminaries of relevant approaches, and outlines data preprocessing and evaluation methods. Chapter 4 explores the main dataset, AMD, providing a brief understanding of the dataset's distribution. Additionally, this chapter encompasses the data preprocessing steps, including balanced sampling, feature selection, and a concise analysis of the k-Means results, to enhance the accessibility of the data for the models.
Chapter 5 includes experiments and hyperparameter tuning conducted on each of the three algorithms. The chapter discusses the output of the algorithms, considering both quantity and quality. Furthermore, in Chapter 6, interpretable results from the algorithms' output are displayed and compared, along with the evaluation results. Chapter 7 elaborates on the implications and limitations of the methods employed. In the final Chapter 8, a comprehensive conclusion is drawn regarding the achievements of the research and also proposes potential directions for future research.

Chapter 2

Related Work

In recent years, with the rapid growth in demand for Android applications, a variety of malware has appeared alongside the release of new applications. Since the appearance of the first DroidDream malware [14], the number of profitable newly born malware has significantly increased. After conducting investigations, the researchers discovered that DroidDream had the ability to exploit a weakness in the Android operating system, allowing it to infiltrate and gain root access to infected devices. Once it had achieved this elevated level of access, the malicious software was able to execute its nefarious operations, including downloading additional code, exfiltrating sensitive information, and carrying out other malicious activities. These findings serve as a stark reminder of the importance of implementing robust security measures to safeguard against Android malware threats.
In this chapter, we first (Section 2.1) discuss the detection of Android malware, including the types of features from malware detection, and the popular detection methods employed. Secondly, we dive into Section 2.2 which addresses how the integration of explanatory malware detection contributes to enhancing the efficacy of conventional detection and clustering tools. Furthermore, we explore the evaluation of interpretative methods following the implementation of malware technologies, assessing their effectiveness. Finally, we present a comprehensive conclusion that highlights the existing gaps and elucidates our innovative approach to addressing these challenges.

2.1 Android Malware Detection

Feature Type. Malicious code features can significantly impact the detection of malware and the accuracy of detection results. There are various divisions in the types of malware detection features. These features can be categorized into three types: static, dynamic, and hybrid features. The static features are extracted without running the application itself. The dynamic features can only be observed when the Android application is executed.

And the hybrid features are a combination of both static and dynamic features [16]. It's also suitable for classifying the corresponding detection analysis methods.

Static features can be obtained without executing the programs. For the Android system, the primary focus is on the application package (APK) file, which is a package document format that instructs the Android operating system to install applications on devices. Numerous valuable features can be extracted from it, such as permissions (requests from applications to access specific actions or data on devices), API calls (to observe whether the malware performs actions that gain access to various functions in the Android platform), package names, related libraries, and application components (e.g., activities, services). Yerima et al. [127] utilized API calls, (Linux/Android) command sets, and permissions in ensemble learning and static analysis, achieving a detection accuracy of 97.3% - 99% with lower false positive rates. Zhu et al. [134] extracted and employed permissions, sensitive APIs, monitoring system events, and permission rates to implement an ensemble Rotation Forest for detecting malicious behaviors, showcasing distinct advantages compared to the Support Vector Machine model.

Dynamic features are revealed in a "living" environment for Android, which means obtaining the features during runtime [47]. Commonly used dynamic features include network communication (network activity generated from the Android applications), system calls (the interface between the user and the kernel also means the requests that need to go through system calls before using any hardware resources [64]), resource usage (e.g., CPU usages, memory, etc.), and file access. Taniya et al. [17] introduced a system to capture system call traces of applications while interacting with Android devices. They considered the frequency of system calls from applications as the primary feature set, feeding it into the J48 Decision Tree and the Random Forest. Shree et al. [39] extracted activity-specific features, OS-specific features, and also the generic network features such as HTTP-based features (Malware can receive/send personal information to remote servers) for malware detection. They conducted experiments on 18 applications from multiple malware families and 14 genuine applications, showing a detection accuracy of 95% - 99.9% and efficiency for Android malware detection.

Hybrid features refer to the combination of static and dynamic features. Multiple models are employed to create an ensemble approach that is adaptable to different scenarios in Android malware. Saracino et al. [100] combined static features (permissions, app metadata at the package level) and dynamic features (user activity, important APIs, and system call) to mitigate malicious behaviors across four levels. Their approach successfully prevented 96% of malware instances from a dataset of 2,800 applications.

However, the static analysis approach has limitations in its inability to identify malicious behavior while the application is running [82]. On the other hand, dynamic analysis is

costly due to the requirement of executing Android applications in a sandboxed environment, which leads to slower analysis [112]. Hybrid methods aim to leverage the benefits and weaknesses of both static and dynamic analysis techniques but also highly rely on each scenario requiring multiple experiments [73].

Feature Selection. Current researchers are utilizing the advantages of machine learning to uncover hidden feature relationships among data samples. As a result, researchers are exploring various models using a large number of features to achieve higher accuracy [61]. Even though these features can increase the probability of finding the correct relationship to distinguish between malicious and benign Android applications. At the same time, an excessive number of features can slow down machine learning technologies and reduce the model's performance [42]. Therefore, several feature reduction methods, along with a multitude of data preprocessing techniques, are employed. Feature selection is one vital method that holds value for machine learning models [24].

Chen et al. [27] and Kang et al. [50] utilized information gain to evaluate the generated feature subset. Information gain is a commonly used non-parametric and non-linear measure that is independent of the distribution of datasets. The genetic search (GS) is a search method based on genetic algorithms, aiming to identify the smallest set of features through various generations. Firdaus et al. [36] employed GS to select features from a pool of 106 strings and conducted experiments using various machine learning methods such as Naïve Bayes (NB), functional trees (FT), J48, random forest (RF), and multilayer perceptron (MLP), achieving an accuracy of around 96%. In addition to these approaches, other methods have been studied, such as a class-driven correlation-based feature selection method [48] and a Deep Q-learning-based Feature Selection method [33].

Machine Learning Methods. Recently, to deal with malware threats, researchers have put great effort into analyzing malware and their common characteristics. Machine learning is an important component in malware analysis, and plays a key role in identifying and categorizing new malware samples automatically.

And multiple researchers have conducted comprehensive reviews and surveys regarding Android malware detection methods based on ML models. Alqahtani et al. [6] divided ML technologies into four types of frameworks: SVM, Naive Bayes, Perceptron, and Neural Network, providing detailed algorithms, article citations, and comparisons. They observed that most of the articles used SVM and NB algorithms, achieving high accuracy of 100% with 0.00% false positives; however, the detection rate was only around 85%. Improvements are needed. Kouliaridis et al. [53] identified gaps among multiple machine learning methods, different metrics, models, datasets, and features that complicate comparisons in a broad view. They established a system to organize past research based on the dataset age, analysis type, ML models, and evaluation metrics and followed a converging schema

Table 2.1: Summary of multiple machine learning models in Android malware detection. The first column lists different ML models, the second column lists the primary strengths, and the third column lists selected weaknesses. The last column provides references that utilize these ML models as their focused researched algorithms in their works. The division of machine learning and part of the pros and cons are referenced by Liu et al. [61]

Models	Advantages	Disadvantages	Reference
Decision Trees (DT)	easy to understand and analysis	potential to overfitting	[136, 111]
Naive Bayesian (NB)	lightweight to gain result	assume the features are independent, which is not the usual case	[103, 2]
Linear Model (LM)	fast and direct	failed to deal with high dimensional data	[95, 94]
Support Vector Machine (SVM)	improved in coping with small, high-dimensional and non-linear problems	sensitive to missing data and data preprocessing	[4, 89]
K-Nearest Neighbor (KNN)	small cost on parameter tuning	computation cost is huge	[120, 8, 126]
Neural Networks (NN) Deep Learning (DL)	high accuracy and fault tolerance	requires huge dataset and computational resources, parameter tuning is laborious	[114, 121, 49]
Ensemble Learning	smarter than single model	expensive to train and requires multiple experiments	[61, 127]
Online Learning	great real-time performance and lower hardware requirements	models are limited to online training	[75, 74]

as a baseline to guide future research.

Table 2.1 [61] presents a division of various ML technologies used in Android malware detection, along with their respective pros and cons.

Some researchers investigate how to improve the algorithm itself. There is a large number of published studies on machine learning in malware classification. A mixed-method approach was employed [68] using lightweight machine learning with app permissions to detect malware, followed by a code analysis to peek into malicious behavior. One decently accurate approach that experiments have shown is AndroDFA [66]. Also, several systematic studies of malware categorization in unsupervised machine learning have been undertaken. Arvind et al. [63] proposed the SOMDROID framework, which involves feature ranking approaches and the Self-Organizing Map (SOM) algorithm and completed a comprehensive evaluation along several clustering and classification metrics. Kanti et al.

[96] introduced Kernel k-means based on k-means clustering methods to prevent the separations from clusters in vector space. Swathi et al. [80] applied Hidden Markov Models as the basis to provide scoring to the k-means model. And different from other research, they examined the accuracy by taking into account the interplay between the dimension and the number of clusters. Aiman et al. [99] also conducted an investigation into the potential use of clustering techniques in malware detection. Their evaluation, which encompassed 18,174 application files, demonstrated the effectiveness of permission features, as well as the amalgamation of AndroidManifest files and application information

2.2 Interpretable Explanation Approaches

In recent years, there has been a proliferation of methods aimed at providing explanations for black-box machine learning models and making their outcomes interpretable in various domains, including identification, estimation, inference, and judgment. These methods aim to make complex models comprehensible to individuals by employing techniques such as LIME (Local Interpretable Model-Agnostic Explanations) [90] or more sophisticated models. The foundation of these models is rooted in various theoretical methods, including Linear Regression [3], Logistic Regression [59], Decision Trees [26], Random Forests [43], Naive Bayes [107], and Gradient Boosting [10].

In these researches, there have been notable contributions toward the development of effective interpretable Android malware detection systems. For instance, Alani et al. [3] proposed a lightweight solution that leverages recursive feature elimination (RFE) to distill the most vital features for distinguishing between benign and malicious software. Furthermore, this system employs Shapley additive explanation (SHAP) values to offer explanatory insights into the selected features, achieving an impressive accuracy rate exceeding 98%.

Similarly, Arp et al. [8] introduced the DREBIN approach that harnesses static analysis and machine learning techniques such as Support Vector Machines to identify crucial features for malware detection. The resultant explanations are highly accurate, with a success rate of 94% and negligible instances of false positives. Additionally, DREBIN provides a valuable resource in the form of a commonly used database for Android malware research.

Martina et al. [70] conducted experiments and discovered the high accuracy from trigrams of systems calls results. They propose a model that combines enhancements to the main model, such as Random Forest, with an auxiliary model like SHAP. The combination methods are based on the degree of disagreement between the main model and the auxiliary model. The obtained results are promising, as the models demonstrate no bias, and

the extracted features are found to be intuitive.

Minami Set al. [104] tackled this challenge through the application of natural language processing techniques, utilizing FCGAT methods for representing function features. Additionally, the attention mechanism is employed to emphasize the importance of functions in the ranking of malware features.

Iadarola et al. [45] employed the cumulative heatmap technique for Android malware detection and identification. This approach leverages the advantages of convolutional neural networks (CNN) while utilizing visualization results to guide classifier decisions and enhance understanding of why intricate network learning produces such models, thereby improving reliability. The interpretability aspect of this approach is applicable to various ML models.

Yajamanam et al. [123] utilized the gist descriptor, which represents image features, to distinguish differences among malware families. One advantage of this method is its ability to avoid the costly extraction of gist features during training or scoring, as the neural network automatically handles this process.

Ni et al. [77] proposed MCSC (Malware Classification using SimHash and CNN), a technique that generates images from disassembled malware codes and feeds them into a CNN using SimHash technology. This method achieves favorable results even when dealing with unevenly distributed samples from different malware families. Furthermore, it exhibits exceptional speed, taking an average of only 1.41 seconds to recognize a new sample.

Jeffrey et al. [30] took a different approach to interpretability, moving away from classification or clustering. Instead, the focus is on understanding the behavior of malware. The distinguishing factor lies in leveraging the MITRE ATT&CK® ontology. This paper automates a method to identify Tactics, Techniques, and Procedures (TTPs) within a sub-part of the control flow graph, which determines the execution flow of a malware executable. By analyzing this specific aspect, it aims to shed light on the malicious behavior exhibited by the malware.

Martin et al. [52] also introduced a novel method using CNN to identify the location of Android app's opcode sequence in malware detection, and employed LIME to explain the importance of this feature.

Though research has been done on interpreting the underlying reasons of XAI models, there still remain some suspicions. Capuano et al. [22] discovered that the explanations are more intrinsic explanation rather than post-hoc (such as SHAP and LIME). Fan et al. [32], on the other hand, raise concerns about the trust level of output interpretations from XAI models. They propose different multiple angles to interpret the results. To address the question of why we should trust our interpretation model, they present a set of guide-

lines for assessment and three critical evaluation methods, including stability (assessing the model's ability to maintain consistent explanation for similar pre-trained models), robustness (measuring the similarity of interpretability results for similar instances), and effectiveness (observing the changes on predictions results when removing several critical features). Nadeem et al. [22] studied existing research on explanation models for security problems from a broader perspective. They discovered a lack of clear separation between users and designers when designing models, as well as a reduced focus on model and explanation verifications. Furthermore, the obtained results often vary across different explanation models. Moreover, Nadeem et al. noted that adversaries could potentially utilize explanations to compromise confidentiality, integrity, and the availability of the model.

Evaluation of XAI Methods for Android Malware Detection. After conducting experiments on machine learning and obtaining the results of interpretation, it is crucial to notice the correctness and employ a metric to assess the algorithm's performance on explanation, and also its effectiveness among other algorithms. The primary objective of the evaluation is to provide evidence to determine whether the model effectively fulfills its objectives [85]. Liu et al. [62] discovered that many researchers tend to focus only on evaluating the XAI results based on over-optimistic classification performance, neglecting the actual malicious behaviors that may result in poor reliability.

For previous works on evaluation methods, some of them require manual evaluation, either by experts or companies. For instance, Van et al. [113] designed an evaluation framework to enhance the effectiveness of users' experiments by utilizing results generated from interpretation. Kim et al. [51] employed a system tool that allows users to test the classification results using user-defined concepts, addressing the challenges in evaluating deep learning interpretation results. However, these evaluation methods through human significantly increase the human cost [58]. Furthermore, human opinions are prone to biases that can violate the scientific research principles of objectivity [19].

Currently, researchers have proposed several quantity metrics for evaluation on classification tasks, including accuracy score, precision, recall, and F1-score [87, 40, 62]. Additionally, execution time (computation cost) is another factor to evaluate the training duration of machine learning methods [58]. For some researchers, the evaluation of interpretation results also remains questionable. Therefore, they have suggested several technologies and established certain rules to assess the evaluation results derived from interpretation. Yang et al. [124] proposed principles to evaluate generalizability, fidelity, persuasibility, etc., encouraging researchers to test their results from these three perspectives. Mohseni et al. [69] proposed a comprehensive framework that combines various elements to provide a detailed evaluation of XAI methods. This framework encompasses multiple dimensions, including the assessment of mental models, end-user satisfaction and usefulness, user

trust and reliance, human-AI task performance, and computational measures. However, these metrics tend to be overly general and lack practical guidance [9]. For deep learning models, notable for hard interpretation ability, Alexander et al. [115] proposed criteria for comparing and evaluating interpretation results, which are based on the accuracy of interpretation and security-focused aspects (completeness, efficiency, and robustness). These criteria are tested on six popular explanation models, leading to the discovery that two models (Integrated Gradients and LRP) achieve the best performance in meeting all the requirements. The authors also observed that in many scenarios trained neural network models have the potential to be overfitted to certain datasets, highlighting the necessity of obtaining more general applicability in deep learning models. In conclusion, evaluation is a challenging task that requires balancing both objective and subjective considerations [41].

2.3 Conclusion

In a nutshell, for the domain of Android malware detection, researchers primarily focus on supervised machine learning techniques, specifically classification tasks. For the explanation methods, the recent advancements in deep learning have garnered significant attention, leading researchers to shift their focus toward these approaches. However, utilizing deep learning networks for explanation purposes poses challenges, such as the extensive computational resources required for generating results and the difficulty in interpreting network outputs. For this reason, the reliability of these network-based technologies for analysis raises concerns.

Therefore, there is a noticeable gap in the application of XAI methods that discuss the performance of clustering techniques in malware family detection and categorization. And also for the evaluation after performing interpretation algorithms, each research study tends to have its own preferences and testing methodologies. In this thesis, we employ a more subjective approach to handle the evaluation results from XAI clustering result.

Chapter 3

Methodology

In this research, our primary objective is to investigate the performance of XAI clustering methods in the categorization of Android malware families. This chapter provides a detailed introduction to the XAI clustering algorithms. We selected three algorithms as the focus point of our work. The main algorithm under investigation is `ExKMC` (Section 3.1). Additionally, we include the baseline model `CART` (Section 3.2) for comparison purposes, and a state-of-the-art model `ICOT` (Section 3.3). The rationale behind our selection has elaborated in Section 1.2.

3.1 ExKMC

`ExKMC` (Expanding k-Means clustering) is first brought up from [38], which is designed for low-cost clustering methods and the tradeoff between explanation and accuracy. `ExKMC` includes two essential input parameters, including k and k', a set of points $\chi = \mathrm{x}^1, \ldots, \mathrm{x}^n \subseteq \mathbb{R}^d$ with cardinality $|\chi| = n$ (n is number of leaves of `ExKMC` tree). The algorithm has two distinct phases, each characterized by its own objectives and strategies.

In the initial phase, `ExKMC` undertakes the construction of a tree consisting of k leaves, employing the famous IMM algorithm [72]. Then, `ExKMC` proceeds to the second phase, marked by minimizing the surrogate cost. Within this phase, `ExKMC` expands the original tree, incorporating additional leaves beyond the initial k. At each step of the iterative expansion, the tree undergoes modifications. Notably, this expansion process involves the partitioning of data using an increasing number of thresholds, thereby leading to more granular divisions.

The mission of algorithm `ExKMC`is the identification of the optimal feature-threshold pair for inclusion, achieved through the refinement of the centers. To elucidate the principle guiding the division process, `ExKMC` introduces a designed surrogate cost. After constructing the initial tree with k leaves (which correspond to the k centers generated by the IMM

algorithm), ExKMC expands the tree with an overall k' leaves (increasing $k' - k$ leaves), adhering to the principle of minimizing the cost associated with the k reference centers. Ultimately, the cluster assignments are determined based on the selection of the best reference center.

A set of k clustering references is generated using the k-Means algorithm. The rationale for optimizing the actual cost is that the original k-Means cost is time-consuming and challenging to recalculate the distances between the nodes and clustering centers. Instead, a surrogate cost is defined as the sum of squared distances between the nodes and their closest reference center.

Definition 3.1 (Surrogate cost)**.** *For centers $\mu^1, ..., \mu^k$, tree T has the clustering $(\widehat{C}^1, ..., \widehat{C}^{k'})$,* the surrogate cost is

$$\widetilde{cost}^{\mu^1,...,\mu^k}(T) = \sum_{j=1}^{k'} min_{i\in[k]} \sum_{\mathrm{x}\in\widehat{C}^j} \left\|\mathrm{x} - \mu^i\right\|_2^2$$

With the surrogate cost, the utilization of the ExKMC algorithm becomes feasible, as outlined in Algorithm 1. This algorithm requires the following inputs: k' leaves, the dataset $\mathcal{D}$, and the pre-defined cluster number k. By leveraging the IMM algorithm and the standard k-Means algorithm, ExKMC is capable of generating a tree T with k leaves from the IMM algorithm and k reference centers from standard k-Means. Subsequently, ExKMC produces a tree T' with assigned labels.

Algorithm 1 ExKMC Algorithm [38]

Require: $\mathcal{D}$ - dataset, vectors in $\mathbb{R}^d$
Require: $\mathcal{C}$ - Set of k clusters
Require: $\mathcal{M}$ - Tree
Require: k' - leaves numbers
  for all $leaf \in T.leaves$ **do**
    Compute gains of split based on the difference between the surrogate cost with split without storing the best feature-threshold pair and cost improvement in splits and gains
  end for
  while $|T.leaves| < k'$ **do**
    select leaf with maximum gain from split
    find its best reference center for its children μ^L, μ^R
    use μ^L, μ^R to form new children of leaves greedily
    update splits and gains
    delete nonsense leaves
  end while
  return T'

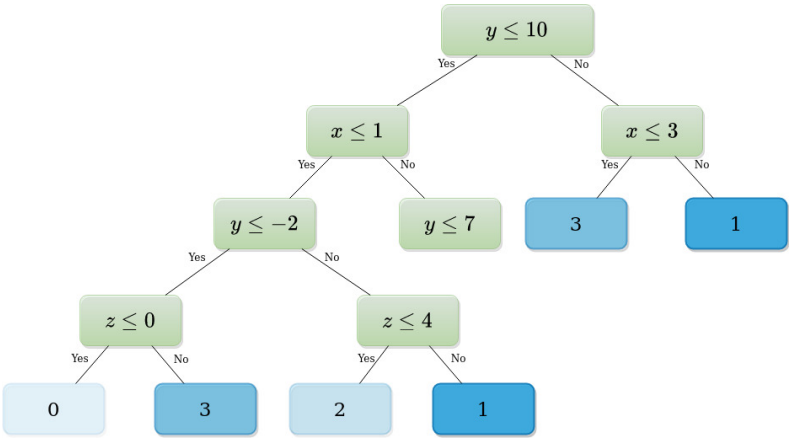


Figure 3.1: Example tree from `ExKMC` result. The basic construction is similar to a binary tree, wherein three axes (x, y, z) are employed to partition the entire dataset of numbers into four clusters, each with a discernible and reasonable path. At each node, the left branch satisfies the given condition, while the right branch follows the opposite.

3.2 Baseline-`CART`

Classification And Regression Tree (`CART`) [55] originated from the decision tree algorithms family, which plays a crucial role in interpretation. `CART` extends the functionality of a decision tree by recursively partitioning the dataset based on features, considering either the Gini impurity or information gain. The resulting binary tree undergoes recursive growth until it reaches the defined limits of tree depth or the minimum number of samples in the leaf nodes. Nodes within the tree signify the features, while branches represent the potential outcomes, and the leaf nodes convey the final results of clustering or classification.

The choice of the `CART` algorithm is driven by several factors. Firstly, it serves as the fundamental algorithm for decision trees, which are highly intuitive and widely used in machine learning research. Secondly, decision trees offer the benefit of generating explanations naturally. Thirdly, the `CART` algorithm is lightweight, characterized by its ability to perform complex computations with minimal resource consumption. Fourthly, in practice, we utilize the labels assigned by the k-Means algorithm as training labels for the `CART` tree. So under the circumstances of using k-Means, we could better investigate the performance of the interpretable part of the whole algorithms compared to `ExKMC`.

Figure 3.2 uses the public and commonly used dataset IRIS [37], and generates an example `CART` tree with 110 samples dataset.

The decision tree starts from a root node, which represents the first decision. The root node divides the data based on the petal width feature in this scenario. If the petal width

is less than or equal to 0.8, the tree continues to the left child node, which predicts the class as "setosa" with high confidence since all the training samples in this branch belong to the "setosa" class. If the petal width is greater than 0.8, the tree moves to the right child node, which further splits the data based on the petal length feature. If the petal length is less than or equal to 4.75, the tree proceeds to the left child node of the current node. In this node, the class is classified as "versicolor" since most of the training samples in this branch belong to the "versicolor" class. On the other hand, if the petal length is greater than 4.75, the tree moves to the right child node of the current node. In this node, the class is classified as "virginica" since most of the training samples in this branch belong to the "virginica" class.

This decision tree provides a visualization of the decision-making process based on the provided features (petal width and petal length) to classify the Iris flowers into different classes (setosa, versicolor, and virginica). It uses if-else conditions at each node to split the data and make predictions. By following the branches and rules of the tree, we can determine the predicted class for a given set of feature values.

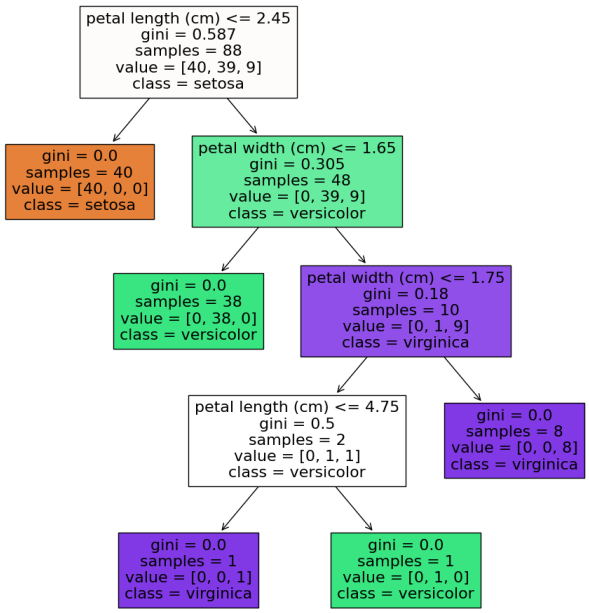


Figure 3.2: Example tree from CART result with IRIS dataset public online.

The Gini index [88] serves as the evaluative metric or optimization criterion in CART for partitioning nodes and determining the subsequent branching paths. It facilitates the assessment of the probabilities associated with node membership across different classes. The equation representing the Gini index is as follows, in which n is the number of

classifications and p_i denotes the probability that a sample being classified for a class:

$$Gini = 1 - \sum_{i=1}^{n}(p_i)^2$$

The objective is to minimize the Gini impurity within the decision tree construction process. The Gini index ranges between 0 and 1, with values closer to 1 indicating a higher degree of randomness in the distribution of instances across classes within a node. Conversely, a value of 0 implies that all samples exclusively belong to a certain class, thus achieving pure classification. An index of 0.5 signifies a uniform distribution among multiple classes.

3.3 ICOT

Clustering via Optimal Trees (ICOT) [15] is built upon the Optimal Classification Trees (OCT) algorithm, utilizing the commonly used clustering metric, the Silhouette Metric. By combining both intra-cluster density and inter-cluster separation, the loss function in ICOT prevents the need for complex considerations regarding tree modification.
ICOT adopts the mixed-integer optimization (MIO) framework to construct the tree globally. While OCT is traditionally used for supervised learning tasks, ICOT leverages insights from the MIO framework to address unsupervised learning tasks with certain modifications.
The loss function definition is based on Silhouette Metric [91].

Definition 3.2 (Silhouette metric)**.** The Silhouette measures the distances between observations in the same cluster to the distance from its second closest cluster. For each observation i, the formulation is as follows, a_i is the mean intra-cluster distance between i and its companion in the same cluster, and b_i is the mean nearest-cluster distance for i and other points in the second closest cluster.

$$score_i = \frac{b_i - a_i}{max(b_i, a_i)}$$

Silhouette Coefficient lies between $-1 \leq score_i$. If $score_i$ is near 0, the cluster i is not distinguished from other clusters, if $score_i$ is near 1, the cluster i can be well clustered, and if $score_i$ approaches -1, it could have a different or wrong clustering result.

The pseudocode of ICOT is listed in Algorithm 2. ICOT begins with a greedy tree and then expands the leaves using a local search procedure until the clustering convergence threshold of the cost function metrics is reached. The original greedy tree can take multiple forms with multiple nominees, and the tree modifies itself recursively by adding leaves.

3.3. ICOT

ICOT returns the tree with the highest clustering metrics. The local search procedure is responsible for assigning the clustering result. The principle of adding, deleting, or replacing is determined based on whether it improves the objective from its current value. All nodes are reconsidered in a list once an improvement is found.

Algorithm 2 ICOT Algorithm [15]

Require: $\mathcal{D}$ - dataset, vectors in $\mathbb{R}^d$
Require: $\mathcal{C}$ - Set of k clusters
  Create a initial greedy tree T, cluster $\mathcal{C}$ and loss l_0
  Potentials to search : S = 1,..., S; loss: $l = l_0$
  while S not Null **do**
    for all $s \in S$ **do**
      if C_k is leaf **then**
        select best new division with loss $\widehat{l}$ ▷ Loss is based on initial tree and calculated by Silhouette Metric
      else
        find the best-modified fit via a different division or division deletion, with loss $\widehat{l}$
      end if
    end for
    if $\widehat{l} < l$ **then**
      Update T' and add all leaves to S. $l \leftarrow \widehat{l}$
    else
      delete s from S
    end if
  end while
  return T'

Chapter 4

Data

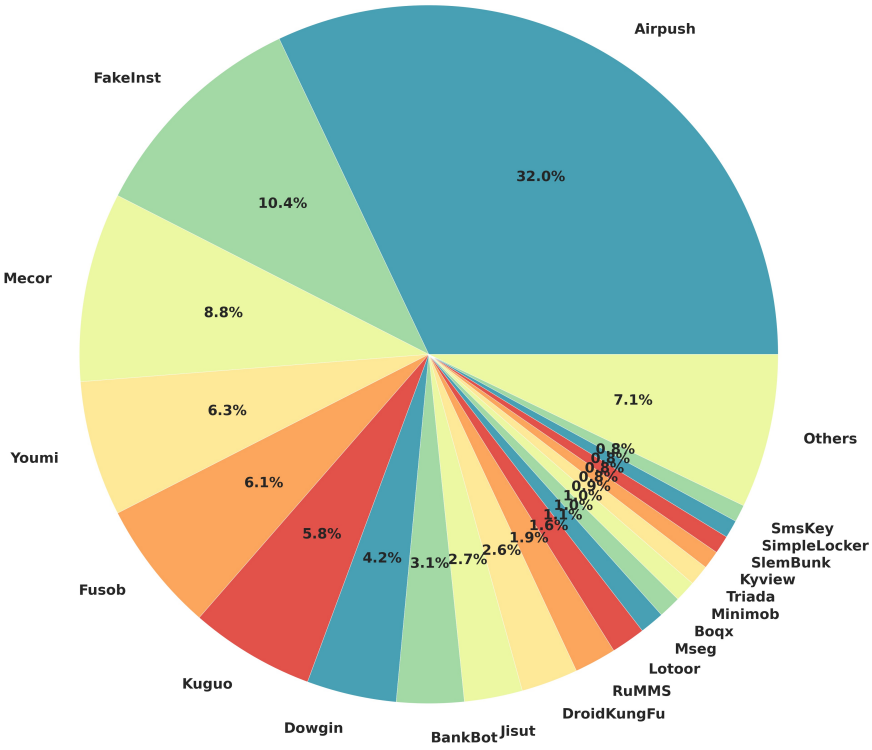
The dataset, AMD, used in this thesis was collected in 2017 by Wei et al. [116]. It contains samples that were observed in the period from 2010 to 2016. It was collected through prevalent anti-virus scan outputs from third parties and compiled using automation technologies. It comprises 24,650 malware applications categorized into 71 families. In addition to offering an extensive collection of Android malware families, AMD also provides detailed information on malware behaviors, which will be useful for later comparisons with the algorithm's generated explanations.

There are several prevalent malware datasets available within the current research domain, including *Circa* (2011, created by Android Malware Genome), *Drebin* (2014, created by Arp et al. [8]), and *AndroZoo* (2016, [5]). Among them, AMD offers distinct advantages as it specifically focuses on details of malware behavior. Furthermore, AMD is relatively up-to-date, covering the time period that corresponds to the emergence and proliferation of various malware families. This is why AMD was chosen as the primary dataset for this research.

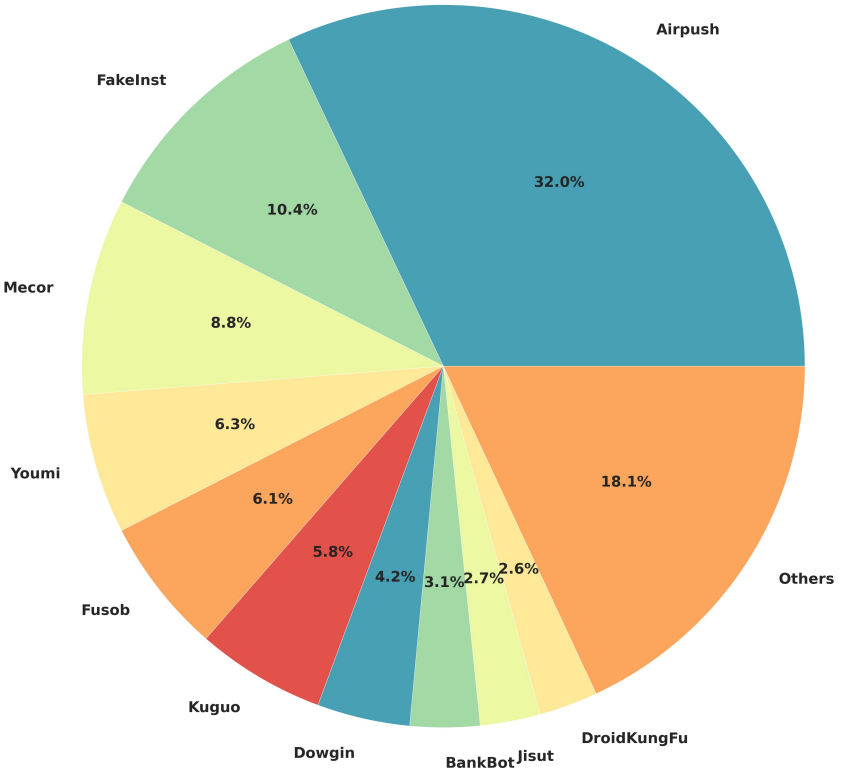
In this section, an initial data exploration and data preprocessing are provided. Data preprocessing also comprises three parts: sample balance, feature selection, and k-Means results.

4.1 Data Exploration

As mentioned above, AMD comprises 24,650 samples and covers 71 families. When considering these 71 families, there are large families like Airpush, which have 6652 malware samples, and small families with fewer than 10 malware samples. Figure 4.1 illustrates the proportion of each family within the entire dataset. The families with sample sizes smaller than 164, which do not fall within the top 20 datasets families, are grouped together and represented as "others" in the figure.



(a) Pie chart of top 20 families based on counts



(b) Pie chart of top 10 families based on counts

Figure 4.1: Family counts in AMD dataset

In detail, the pie chart provides a clear and intuitive representation of the relative proportions of each family within the dataset. The size of each slice reflects the family's relationship to the entire dataset and its contribution to the whole. Referring to Figure 4.1, we can observe that the largest family, Airpush, occupies approximately 32.0%, which is almost one-third of the entire dataset. From the family FakeInst to the family Kuguo, they collectively account for over 5% of the whole dataset. The relatively smaller families make up around 3%.

From Figure 4.1a, we can see that the top 20 families and their samples constitute approximately 92.9% of the dataset, while the remaining datasets contain less than 164 samples. Furthermore, in Figure 4.1b, the top 10 families and their samples account for approximately 81.9% of the dataset. It is also worth noting that the largest family takes up the highest proportion within the dataset.

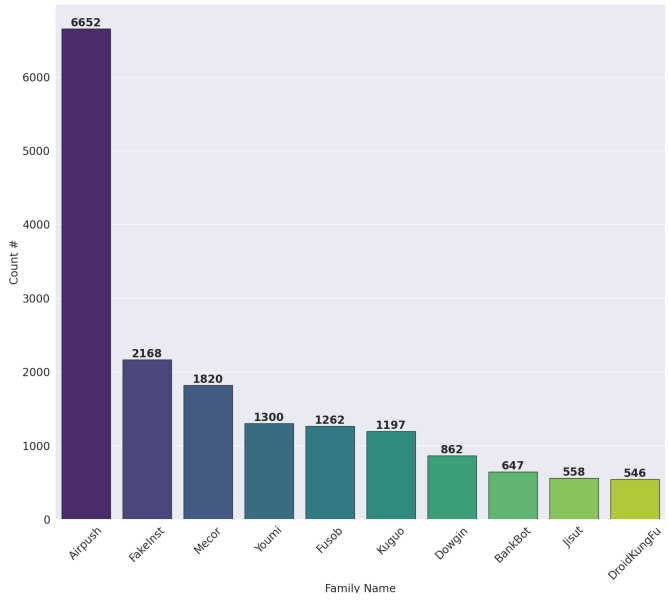


Figure 4.2: Histogram displaying the distribution of the top 10 families in the AMD dataset in descending order, along with their respective counts.

From Figure 4.2, it is evident that Airpush comprises approximately one-third of the entire dataset, amounting to 6,652 samples. Comparatively, the 10th largest family, DroidKungFu, consists of 546 samples, resulting in a significant difference of 6,106 samples. Therefore, it is necessary to implement a sampling process to address the imbalance, particularly concerning the Airpush family. Sample distribution plays a significant role in clustering tasks, as highlighted in [128]. Inspired by Zhao [131], for simplicity and better representation of results, we select the top 10 families as the experimental data.

4.2 Data Preprocessing

For unsupervised machine learning, k-Means, `ExKMC`, and other interpretable algorithms are commonly used. Data preprocessing is necessary and beneficial, which includes techniques such as feature scaling, feature selection, and sample balancing.

The benefits of data preprocessing are numerous. Some unsupervised machine learning algorithms rely on distance-based calculations [108]. Feature scaling techniques like standardization or normalization ensure that all features are on a similar scale. This prevents features with larger magnitudes from dominating the analysis and ensures that the algorithms consider all features equally [133]. Balancing the skewed dataset is also crucial as unsupervised learning algorithms are highly sensitive to imbalanced datasets, which can introduce bias even in large datasets.

The evaluation metric used for assessing the k-Means results is MoJoFM. MoJoFM is one of the clustering measures used to evaluate clustering results. It was introduced by Zhihua Wen and Vassilios Tzerpos in 2004 [117]. MoJoFM is an effective measure based on the Minimum Overlap (MoJo) distance, which is also utilized by Zhao in comparing clustering results for malware detection [131].

The similarity metric can be used to select the most similar cluster for each cluster result. The Silhouette score is employed to assess the quality of clusters in typical clustering analyses. Additionally, MoJoFM can be utilized to provide a quantitative evaluation of the algorithm's performance.

MoJo distance (Minimum Overlap distance) forms the foundation of the MoJoFM metric, which measures the similarity or difference between two clusters. For instance, given two clusters A and B, MoJo distance calculates the minimum number of "Move" and "Join" operations required to transform A into B or vice versa. Here, A represents the clustering generated by the algorithm, and B is the reference or "gold standard" labeling [117].

The equation below describes a general rule for calculating the MoJoFM result, where M represents the algorithm, A denotes the algorithm's output, B represents the "gold standard" and n is the number of clusters. $mno(A, B)$ is how close A comes to B, which defines in its original article and calculates the actual maximum distance to partition B for the denominator of its formula [117].

$$MoJoFM(M) = (1 - \frac{mno(A, B)}{\max(mno(\forall A, B))}) \times 100$$

4.2.1 Sample Balance

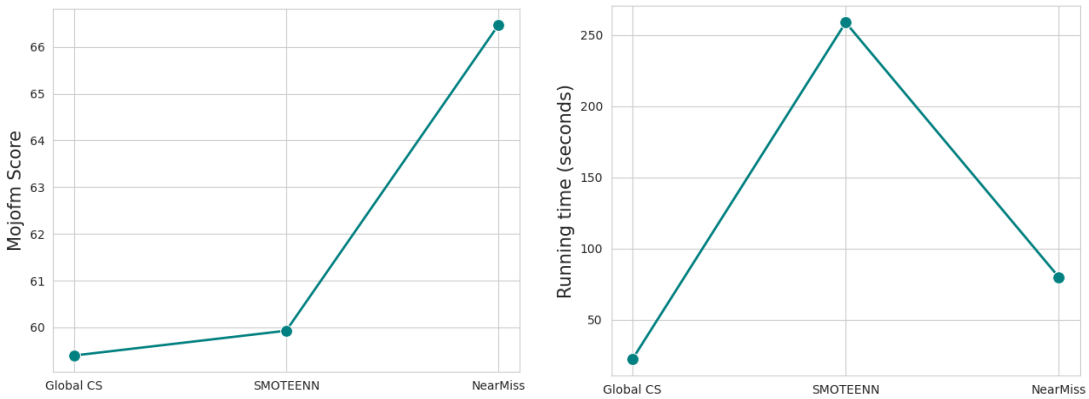
To deal with the unbalanced data distribution, various preprocessing methods are used, including over-sampling the minority class [106, 81] or under-sampling the majority class

[83, 18]. `NearMiss` is our selected under-sampling method. And in this thesis, the focus is on the first method (`NearMiss-1`), which aims to identify and reduce the number of significant negative examples that overshadow the positive examples in the minority class. The selection strategy is based on the top three closest positive examples, aiming to minimize the average distance between the negative examples and these positive examples.

Applying under-sampling techniques offers several advantages. For example, if the majority class (negative labels) constitutes 80% of the dataset, while the minority class (positive labels) constitutes only 20%, employing under-sampling would preserve the information in the minority class by maintaining the same number of instances as before, or even improving it. This helps to prevent the loss of meaning that can occur when the majority class dominates the dataset. However, it's important to note that if a rigorous threshold is applied for removing features, some important features in the negative labels may be lost, potentially causing issues in clustering.

`NearMiss` [65] employs a selection strategy where it identifies instances from the majority class that possess the lowest average distance to the three nearest instances belonging to the minority class.

Therefore, we utilized the `NearMiss` algorithm. It is essential to explore different sampling algorithms to determine the most suitable one for a given dataset. Different sampling methods can have varying impacts depending on the distribution of the dataset. The evaluation metric used for comparing different sampling methods is crucial. We have experimented with oversampling methods such as `SMOTE` [25], `Global CS` [35], as well as the combined over-sampling and under-sampling method called `SMOTEENN` [12].



(a) MoJoFM score on three sampling technologies

(b) Running time on three sampling technologies

Figure 4.3: MoJoFM score and running time of different sampling technologies on AMD dataset

Then the result from MoJoFM is obtained by comparing the k-Means prediction results and the true labels to examine the sampling algorithms' performance. The running time and the MoJoFM result are displayed in Figure 4.3. When comparing these algorithms, the best result is obtained from `NearMiss`, which is an undersampling method. In Figure 4.3a, `Global CS` and `SMOTEENN` display the not great result with only a round 60% MoJoFM score. As for `NearMiss`, the MoJoFM score gives a decent result of approximately 66.5%, which is the highest among all three sampling methods. Additionally, when we consider the running time as shown in Figure 4.3b, `SMOTEENN` is the slowest with a runtime of 250 seconds, while `Global CS` and `NearMiss` perform similarly, taking 80 and 20 seconds, respectively. Consequently, it is evident that `NearMiss` could be the superior method among these three algorithms.

The reason why oversampling and combined sampling methods are not as effective as undersampling is that the oversampled dataset becomes excessively large. For each family, the samples are duplicated to an increased number, 6652 samples. For instance, after applying the `SMOTEENN` method to each family, the result can be presented as follows: (family cluster No., corresponding number of samples). (0, 6393), (1, 6652), (2, 6609), (3, 6646), (4, 6650), (5, 6652), (6, 6652), (7, 6552), (8, 6652), (9, 6534). Consequently, the overall result is unsatisfactory due to the combination of factors. Firstly, the substantial number of duplicated samples, reaching up to 66,000, might negatively impact the outcome. Secondly, the poor performance in MoJoFM score and running time, as depicted in Figure 4.3.

Therefore, based on our evaluation and comparison of different sampling methods in Figure 4.3, we have selected the `NearMiss` method to continue our experiments. The `NearMiss` algorithm has shown promising results in addressing the challenges posed by imbalanced datasets. By selecting the `NearMiss` method, we aim to further investigate its effectiveness in improving the performance of our machine learning models.

4.2.2 Feature Selection

The AMD dataset comprises 695 features, categorized into the following groups: permission, intent, network activities, telephony, provider, NFC, location, media, bluetooth, app, OS, inputmethodservice, hardware, accounts, and other detailed Android methods. Extensive numbers of features for ML technologies could compromise the outcome, resulting in biases or even poor model performance. Therefore, this thesis also investigates feature selection techniques to reduce the number of Android malware features before conducting experiments on algorithms.

Feature selection [42] finds wide-ranging applications in various scenarios, including pattern recognition [78], artificial intelligence [20, 97], and statistical analysis [31, 23]. All

studies emphasize the significance of relevant features and the elimination of redundant ones to determine the threshold for division or recognition [132]. Feature selection is a process that identifies the most important features to enhance the impact and sustainability of the model, regardless of whether it involves clustering, classification, or regression tasks in machine learning. Additionally, by reducing dimensionality, feature selection aids in better data visualization and comprehension.

Based on the availability of labels in the dataset, feature selection methods can be classified as supervised, semi-supervised, or unsupervised. In the context of this thesis, our focus is on unsupervised feature selection methods. Unsupervised Feature Selection (UFS) is designed for non-labeled datasets.

`FRUFS`[1] is a method designed to address the challenges associated with unlabeled datasets and models. It falls under the category of feature relevance-based Unsupervised Feature Selection (`FRUFS`) and employs supervised algorithms, such as XGBoost, to rank features based on their importance.

The design initiative of `FRUFS` draws inspiration from linear regression, specifically, the equation $\sum_j \alpha_j^k x_j^i$, where α^i represents the importance of the corresponding feature x_j^i. This importance can be interpreted as the weight used to predict the target variable. The primary objective is to utilize a set of coefficients to predict each feature. The equation representing this process is as follows:

$$x_k^i = \sum_j \alpha_j^k x_j^i$$

α_j^k is the j-th feature coeeficient to predict the k-th feautre.

And x_k^i means that we are going to predict the feature importance on k-th feature based on other features of x

$$\begin{aligned} X &= XW^T \\ \Rightarrow X - XW^T &= 0 \\ \min \left\| X - XW^T \right\|_F \end{aligned} \tag{4.1}$$

In equations, $\left\| X - XW^T \right\|_F$ is the Frobenius norm that could get the sum of squares of all values in the matrix.

In a more concrete matrix form based on Equation 4.1, we have the matrix X with

[1] `https://github.com/atif-hassan/FRUFS`

dimensions ($n \times d$), where n represents the number of samples and d represents the number of columns. Our objective is to learn the matrix W through machine learning. Matrix W has dimensions ($d \times d$) and initially contains randomly assigned coefficient values. For instance, considering the first row of W, denoted as W_i^j where i is 1 and $j \in \{1, 2, ...d\}$, the values of W_1^j pertain to the second feature among the d features, and its purpose is to predict the first feature. Similarly, in the first column, when j is 1, the coefficients W_i^1 represent the weights associated with the second feature for all other features.
To obtain the final importance value for each feature, we calculate the average value across all instances i of feature j in X_{ij}. The process of redefining the importance is also referenced from [135]. Therefore, we apply `FRUFS`, the cut-off importance ranking of features is shown in Appendix 8.1.
After applying feature selection of `FRUFS` algorithm, the total number of features decreased from the original **695** to a reduced number of **209** features.

4.2.3 k-Means Result

Before delving into the results of `ExKMC` and the corresponding experiments, it is essential to understand the foundation of `ExKMC`, which is based on the k-Means algorithm. By examining the k-Means results before applying advanced `ExKMC`, we can gain insights into the data's separation and observe how k-Means contributes to enhancing the performance of both the dataset and `ExKMC`. Therefore, in this section, we present an exploration of the pure k-Means results to provide an overview of the data partitioning and the underlying mechanisms that aid `ExKMC` in generating its outcomes.

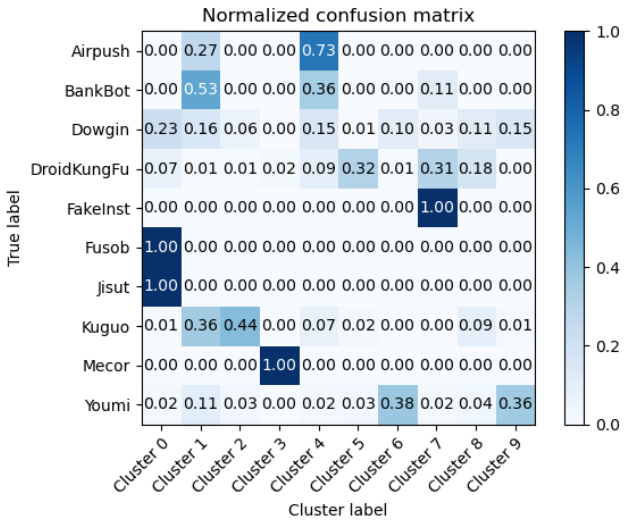


Figure 4.4: Confusion matrix of k-Means on AMD top 10 families after data preprocessing

After applying the k-Means algorithm, the confusion matrix in Figure 4.4 was generated,

targeting 10 classes. The x-axis represents the cluster results obtained from k-Means, while the y-axis represents the true labels of family names. Several important observations can be made from this confusion matrix, which requires careful consideration or discussion after the experiment and obtaining the explanation result.

- The Fusob and Jisut families are difficult to distinguish since they are both perfectly assigned to one cluster, and they have a 100% assignment rate for their true labels
- Similarly, cluster 1 presents challenges in distinguishing different classes. There are three labels that can potentially match the actual class, with matching rates exceeding 25%.
- A similar phenomenon occurs in cluster 4, where the matching rate for the true label "Airpush" is 73%, but it is also 36% for the label "BankBot." This suggests that "Airpush" and "BankBot" may share some common features in some way. Further investigation is needed in subsequent experiments to explore this relationship.

Chapter 5

Experiment

In this chapter, our focus will be on the experiments including the algorithms `ExKMC`, `ICOT`, all conducted on the AMD top 10 families. Firstly, in Section 5.1, we will introduce the hardware and software settings used for these experiments. Secondly, for each algorithm, the experiments will be divided into three main components. The first component presents the experiments themselves. For `ExKMC`, this includes confirming the matching between clusters from the experimental results and the true labels for this multi-label clustering task. This is achieved through a similarity comparison table. The second component involves parameter tuning to select the best parameter combination for optimal outcomes with the given datasets. Next, we explain the results of the explanation tree generated by the algorithms through a cut-off example tree, as the complete tree generated by the algorithm is excessively large. This is also the key step and crucial principle to obtain the explanation result in the next chapter. In the end, we compare the clustering accuracy results of three algorithms, providing a more quantitative perspective to examine their performance.

5.1 Setup

The experimental setup is unified to make an easy reproducibility. The entire experimental code is available online[1], and the descriptions of the experiments are provided as follows:

- Language: Python 3.9 (with Anaconda as virtual environment); Julia 1.1.0
- Hardware and software: Processor 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz × 8 ; Memory is 8 GiB; operation system is ubuntu 22.04; no GPU used

[1] `https://github.com/hhant-max/Thesis`

- Algorithms: ExKMC[2]; ICOT[3]; CART[4]

The experimental results are averaged and interpretation results are concluded based on ten repetitive runs for each algorithm ExKMC, ICOT, and CART.

5.2 ExKMC

After analyzing the family through the k-Means algorithm and visualizing the results, this section focuses on determining the correspondence between clusters and families. Additionally, we explain the generated results from ExKMC and summarize the features with ground truth labels.

5.2.1 Cluster and Label Matching

Since we have identified the top 10 families based on their numbers, we set the value of k as 10 in ExKMC. Consequently, the model automatically generates 10 clusters based on the learning results (e.g., cluster 0 to 9). However, as the results are not directly connected to the true labels, which represent the family names, we cannot confirm which cluster corresponds to which family without a method to evaluate and quantify the similarity between clusters and family labels.

To address this, we utilize a simple and intuitive approach for matching. The results are obtained by comparing each family to the 10 families using the Jaccard similarity coefficient. This method offers a straightforward and efficient way to measure the similarity between clusters and labels. The Jaccard similarity is an intuitive metric commonly used to quantify the similarity between two clustering results. It is determined by calculating the intersection of two sets and dividing it by the union of their sizes. The Jaccard similarity coefficient ranges from 0 to 1, with values closer to 1 indicating a higher similarity between the clustering results. The formula for Jaccard similarity can be expressed as follows:

$$J(A, B) = |A \cap B| \,/\, |A \cup B|$$

It's verified and used by many unsupervised machine learning and clustering results [56, 29, 54, 79].

Table 5.1 presents the results of applying k-Means to Android malware. The first column represents the clusters assigned by the algorithm, ranging from cluster 0 to 10. The values in the second row indicate the Jaccard similarity coefficient corresponding to the samples

[2] https://github.com/navefr/ExKMC/tree/master
[3] https://github.com/agniorf/ICOT-Example/tree/main
[4] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

within each cluster and the respective families. A similarity coefficient approaching 1 indicates a strong match between the cluster and label, while a coefficient of 0 signifies no overlap.

Table 5.1: Jaccard distance between clustering results and the original family samples

Cluster No.	**Airpush**	**BankBot**	**Dowgin**	**DroidKungFu**	**FakeInst**	**Fusob**	**Jisut**	**Kuguo**	**Mecor**	**Youmi**	**Label**
0	0	0	0.0729	0.0214	0	**0.4320**	0.4297	0.0017	0	0.0056	Fusob
1	0.1242	**0.2810**	0.0682	0.0045	0	0	0	0.1737	0	0.0456	BankBot
2	0	0	0.0399	0.0036	0	0	0	**0.3993**	0	0.0183	Kuguo
3	0	0	0.0018	0.0082	0	0	0	0	**0.9785**	0.0009	Mecor
4	**0.4323**	0.1720	0.0670	0.0369	0	0	0.0015	0.0312	0	0.0084	Airpush
5	0	0	0.0040	**0.2986**	0	0	0	0.0149	0	0.0219	DroidKungFu
6	0	0	0.0750	0.0086	0	0	0	0	0	**0.3393**	Youmi
7	0	0.0467	0.0136	0.1416	**0.6825**	0	0	0	0	0.0067	FakeInst
8	0	0	0.0842	**0.1435**	0	0	0	0.0692	0	0.0266	DroidKungFu
9	0	0	0.1104	0	0	0	0	0.0085	0	**0.3129**	Youmi

For each cluster (each row), we calculate the largest Jaccard distance among all possible families and assign this maximum value to confer the label (in the last column). For instance, in cluster 0, we compute the Jaccard distance to 10 potential families, resulting in 6 positive values. Among these values, Fusob has the highest score compared to the others. Consequently, we assign Fusob to cluster 0.
And additionally, based on the findings presented in Table 5.1, we can also derive significant insights from the similarity results.

- The model results reveal two competing families, namely DroidKungFu (in clusters 5 and 8) and Youmi (in clusters 6 and 9). Through a deeper investigation of clusters 5, 6, 8, and 9, we observe that the similarity between clusters 5, 8, and DroidKungFu are 0.2986 and 0.1435, respectively, indicating a gap of perfect matches with a similarity score of 1. The same situation applies to the Youmi family, with approximately 30% similarity in both clusters. These findings suggest that clusters 5, 6, 8, and 9 are closely related and difficult to distinguish from each other.

- The presence of competing families, DroidKungFu and Youmi, also implies the absence of the Jisut and Dowgin families. Through careful observation of the first

row, we note that Jisut has the second highest similarity compared to Fusob. This indicates the difficulty in distinguishing between Jisut and Fusob based on their similarity. These observations align with the descriptions provided in Figure 4.4 (Chapter 4) for Jisut and Fusob. Additionally, identifying Dowgin proves to be a challenging task, which is consistent with the research conducted by Zhao et al. [131].

- Taking a vertical view of the table, we notice that certain families are easily distinguishable from others, as they only have one or two matching clusters. For instance, FakeInst exhibits similarity with only cluster 8, while having a similarity score of 0 with other clusters. Conversely, families such as Youmi, Dowgin, and DroidKungFu present a more complex scenario, indicating the need for further study in this research into their shared features, which might lead to the difficulty in separating them.

- If we would like to rank the similarity matches among the families, based on their similarity values, we can list them as follows: Mecor (0.9785), FakeInst (0.6825), Fusob (0.4320), Airpush (0.4323), Kuguo (0.3993), Bankbot (0.2810), Youmi (0.3393, 0.3129), DroidKungFu (0.2986, 0.1435).

5.2.2 Parameter Tuning

For the ExKMC algorithm, there are two important parameters to control and experiment with k and max_leaves. Since k is settled at 10, we focus on varying the value of max_leaves in our experiments. The surrogate cost is defined as the ratio of k-Means to

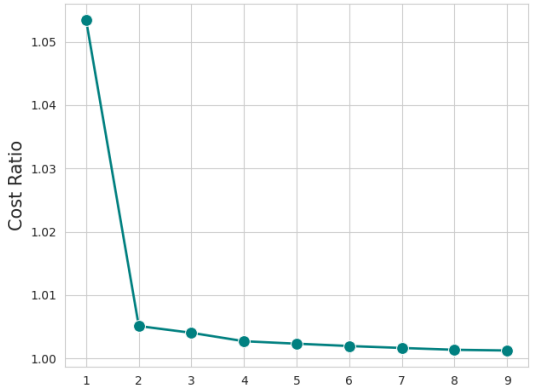
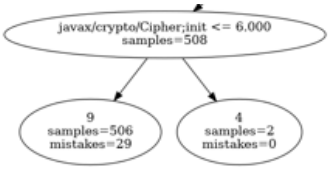


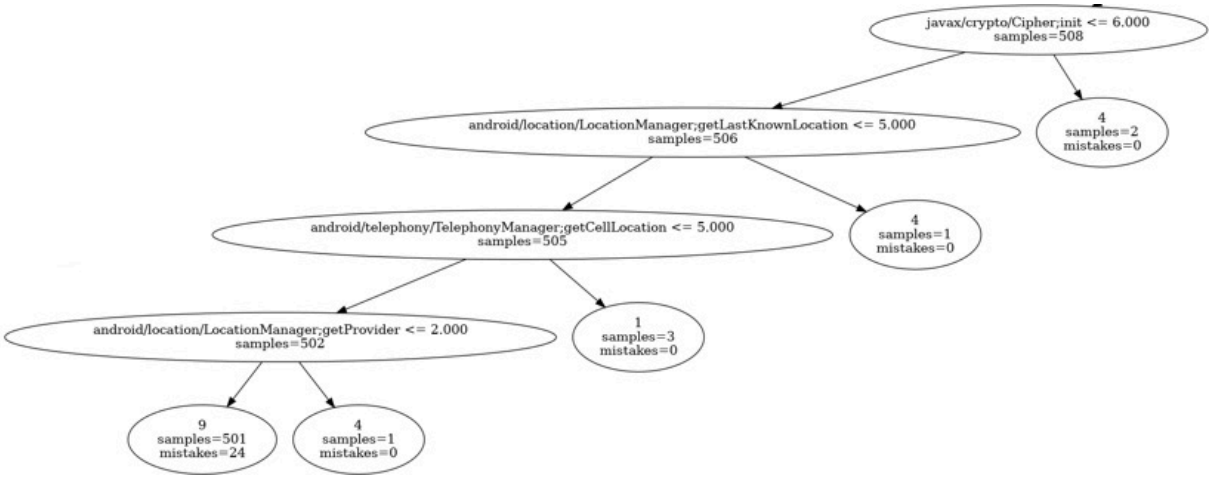
Figure 5.1: Cost ratio of parameter tuning on ExKMC algorithm

ExKMC. When this value converges to 1, it indicates that the interpretable result tree has a similar effect to ExKMC while providing interpretability [38].

Experiments show that as the maximum number of leaves increases and decreases to 1 after $i = 2$, the cost ratio is nearly equal to 1, with only a slight difference as i increases. However, as i increases, the tree naturally expands with more, and possibly abundant, leaves. Therefore, the final parameter selection also needs to take the interpretation result into account, which will be discussed later in the result section. Based on the current analysis, the best results are obtained when the value of *max_leaves* is set to 3, 4, 5, or 6. For values 1 to 3, the results are worse compared to when $i = 4$. On the other hand, when $i = 7$, $i = 8$, or $i = 9$, the resulting tree becomes large and filled with too many branches, which can be illustrated later on.



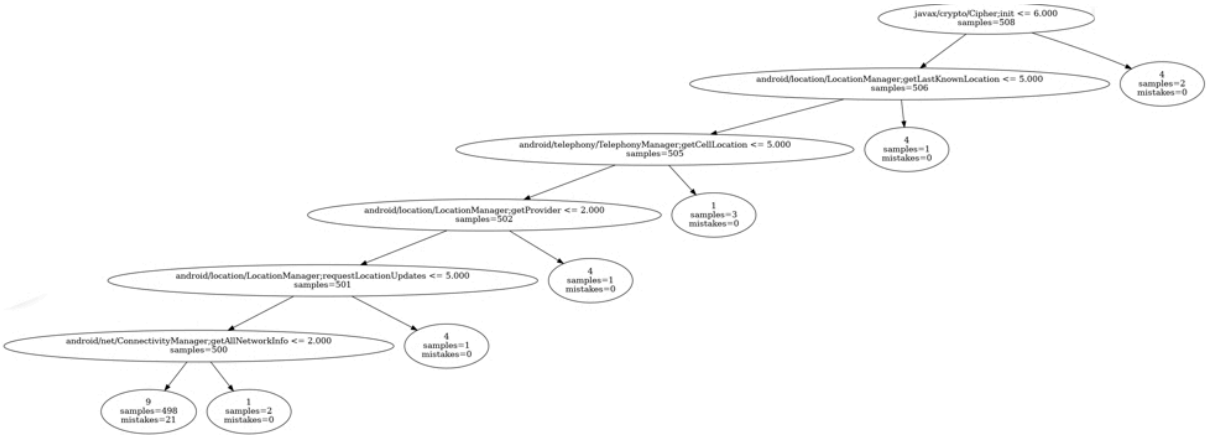
(a) *max_leaves* of 3



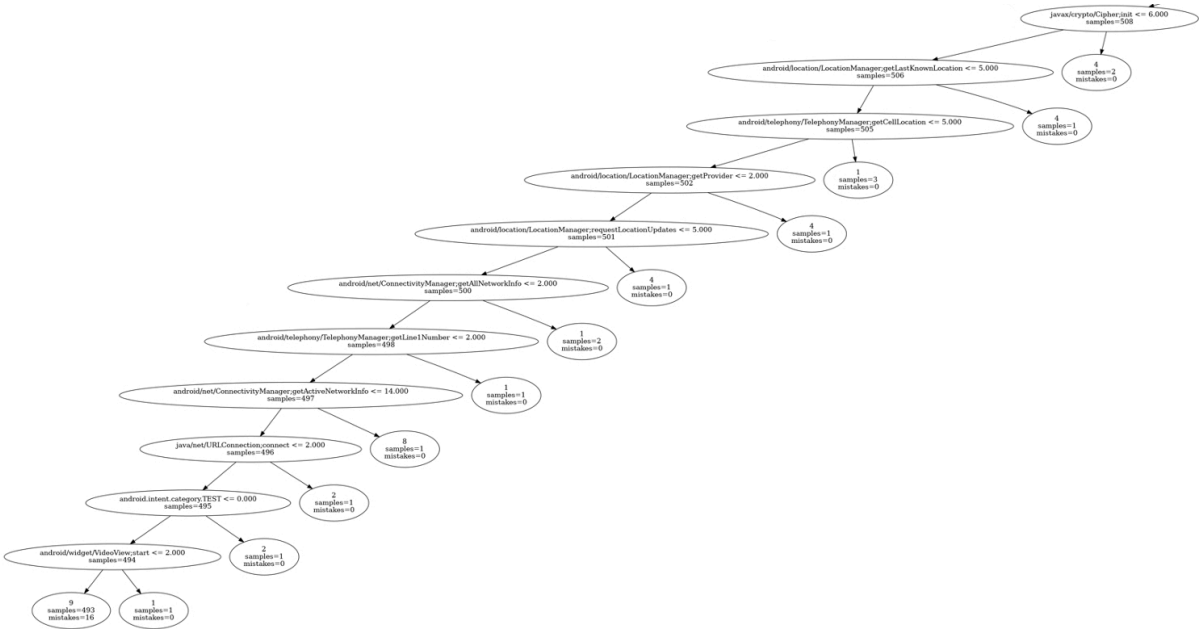
(b) *max_leaves* of 5

Figure 5.2: Example tree of *max_leaves* of 3, 5 from `ExKMC`

For interpretable algorithms, it is vital to observe the generated tree with explanatory results. Since the generated tree is quite large, we only display the partial tree for the selection of the parameter *max_leaves* in Figure 5.3. From the observations, we can see that for the branch of the feature *Cipher;init*, when it is smaller than 6, there are 509 samples separating into cluster 9 with a misclustering count of 29. However, when *max_leaves* is set to $6 * k$, it separates one sample from cluster 4, three samples from cluster 1, and one sample from cluster 4, resulting in 24 mistakes compared to the 29 shown in Figure 5.2a. Furthermore, increasing *max_leaves* to 6 reduces the mistakes to 21, and setting



(a) *max_leaves* of 6



(b) *max_leaves* of 9

Figure 5.3: Example tree of *max_leaves* of 6, 9 from ExKMC

max_leaves to 9 further decreases the mistakes to 16.
Nevertheless, as *max_leaves* increases, the tree becomes excessively large, and the improvement in separation occurs incrementally. Therefore, it raises the question of whether it is worth expanding the tree endlessly despite the increase in clustering samples. Based on the results from Figure 5.1, the final selection for the parameter of the `ExKMC` algorithm is $max\_leaves = 6$.

5.2.3 Result

In Figure 5.4, a comprehensive result is presented from the `ExKMC` generation process. For instance, at the root level, a key feature is identified where the count of *PowerManager$WakeLock;release* is greater than 5. Based on this *criterion*, `ExKMC` assigns the corresponding samples to cluster 6, which represents the Youmi family and consists of 556 samples. On the other hand, samples with a count of *PowerManager$WakeLock;release* smaller than 5 proceed to subsequent branches, awaiting further features to facilitate their separation.
Furthermore, within cluster 6, a notable feature that sets it apart from other clusters is the *PowerManager$WakeLock;release*. This principle is also crucial for identifying essential interpretable features using interpretive clustering methods, enabling the comprehensive determination of features for all families in the final result table in Chapter 6. By comparing the results among different algorithms, insights can be gained into the performance and explanation of these algorithms.

5.3 `ICOT`

In this section, the experiments primarily center on the `ICOT` algorithm. These experiments involve injecting the top 10 families of AMD dataset into the `ICOT` algorithm. Additionally, considering that parameter tuning plays a crucial role in achieving the best adaptable results for each machine learning method, we discuss the process of tuning key parameters in Subsection 5.3.1. Furthermore, we present an outlined explanation tree derived from the complete result, which is then interpreted in Subsection 5.3.2.

5.3.1 Parameter Tuning

There are multiple parameters to tuning for `ICOT`.

- *Criterion*. *Criterion* is used for evaluating the splits by confirming the dense regions as well as sparse regions [15]. Here, the *criterion* is defined as the Silhouette

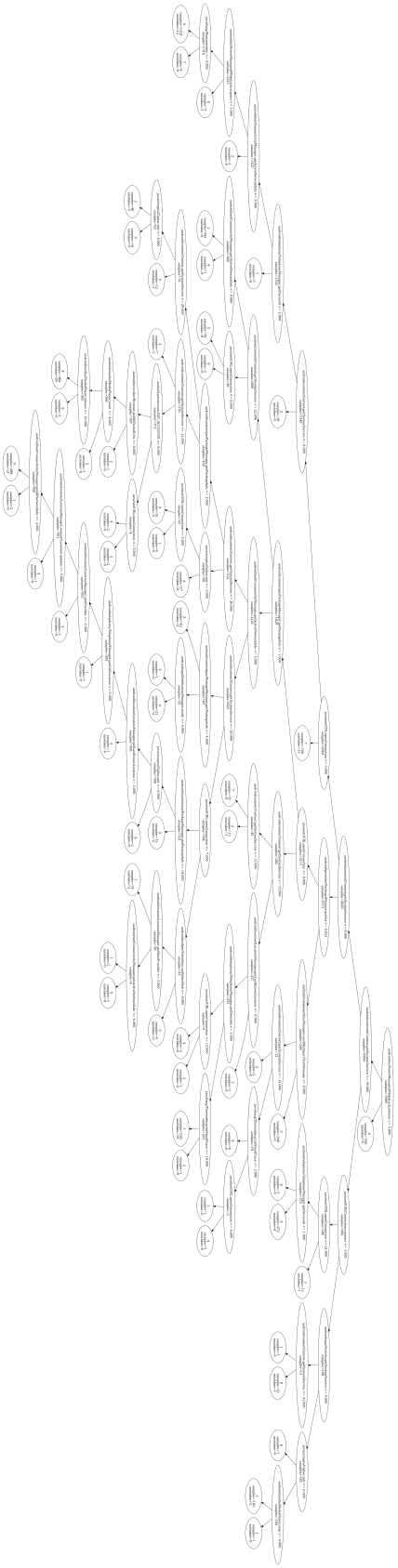


Figure 5.4: Explainable tree generated from ExKMC algorithm

Coefficient score. The Silhouette Coefficient in the ICOT algorithm is from the comparison of two distances: the average distance within a cluster (a) and the average distance to the nearest neighboring cluster (b), both calculated for every sample.

- *Ls_num_tree_restarts.* The *ls_num_tree_restarts* parameter controls the number of random restarts used in the local search algorithm, which corresponds to the initial greedy tree in the ICOT algorithm. Its effect primarily pertains to computational cost, and we have set a default value of 10. Increasing this value leads to a linear increase in the time spent. The complexity parameter plays a crucial role in achieving a balance between performance and the complexity of the tree to control overfitting. We have set it to the recommended value of 0. The value of this particular parameter was determined through subsequent timing experiments.

- *Max_depth.* *Max_depth* is the parameter to control the maximum depth of the fitted tree. After conducting initial experiments and adhering to the principle of maintaining consistency with the ExKMC method, we decided to focus more on *max_depth* for in-depth research. As a result, we will only present the tuning process and effects of the *max_depth* parameter. The complete result on different choice ranges from 4 to 10 of *max_leaves* can be found in Table 5.2.

Table 5.2: ICOT result parmeter

Max Depth	4	5	6	7	8	9	10
Silhouette	0.4750	0.5067	0.5356	0.5408	**0.5453**	0.5447	0.5422
Running time (min)	113	132	172	164	190	212	315

From Table 5.2, we observe a gradual increase in performance from 4 to 7, with the peak achieved at 8. Subsequently, at 8 and 10, the results exhibit similarities and do not demonstrate substantial improvements. This suggests that the maximum number of leaves set to 8 adequately captures the clustering and effectively explains the entire dataset. Consequently, we have selected this parameter value of **8** for the final interpretable result.

5.3.2 Result

After tuning the parameters, we conducted research on the interpretable tree results generated by the ICOT algorithm. The complete result is presented in Figure 5.5.
To provide a better illustration, we have extracted a small portion for discussion, as shown in Figure 5.6. In this figure, based on important features the leaves are segregated

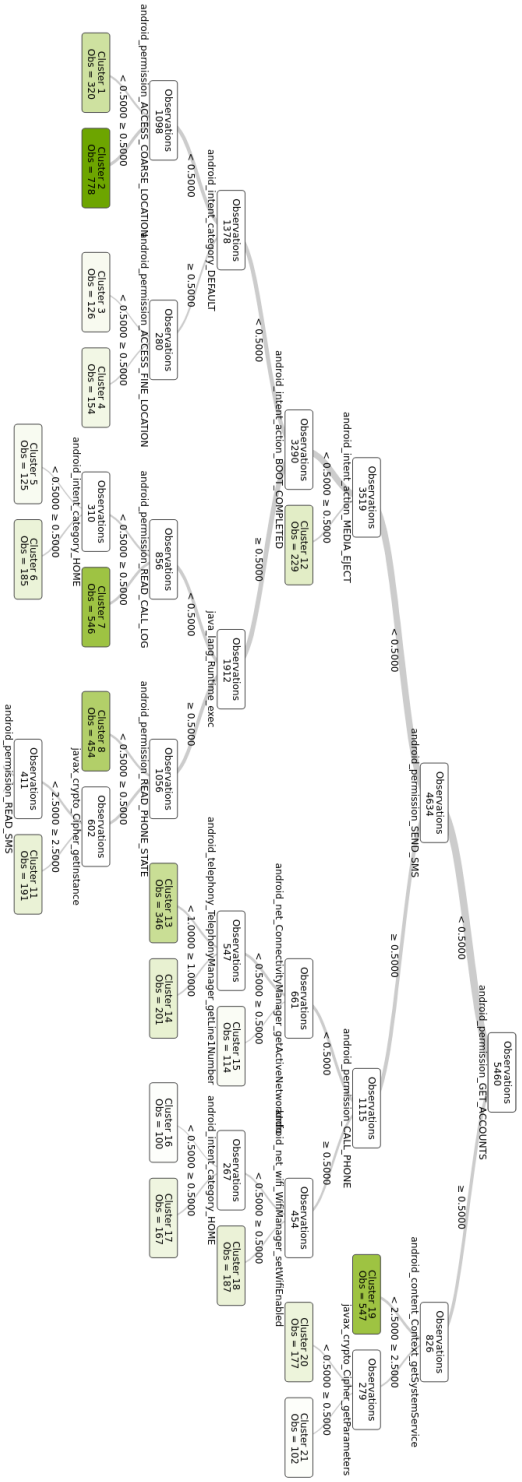


Figure 5.5: Explainable tree generated from ICOT algorithm

into two branches, as depicted in the fundamental visualization. Additionally, the detailed comparisons with the true labels aid in better understanding the performance of clustering, as elucidated in the caption box of the figure, which assigns the predicted probabilities as cluster labels.

At the root node, there are 1056 observations. The feature analysis reveals that the presence of *READ_PHONE_STATE* does not lead to any path to the left branch, which corresponds to cluster 8, since the value of *READ_PHONE_STATE* for the left branch is less than 0.5. and the two possible values for these features are only 1 and 0. The left node cluster 8 represents 98.68% of the total outcome 6, accounting for 440 observations. This suggests that the malware family associated with outcome 6 does not typically exhibit the *READ_PHONE_STATE* feature.

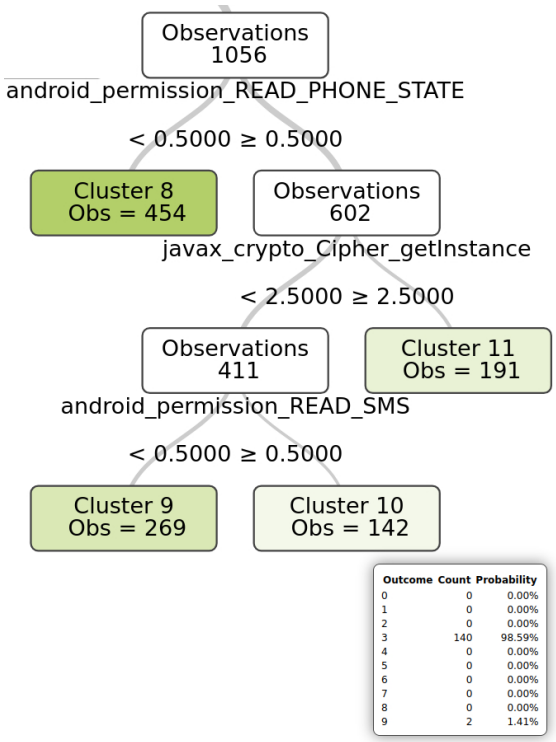


Figure 5.6: Example explainable tree from `ICOT` algorithm

Continuing down the branch, we encounter 602 observations where the relevant feature for cluster division is *Cipher_getinstance*. In this branch, the outcome distribution is as follows: 0 (3 samples), 2 (14 samples), 3 (491 samples), 7 (76 samples), and 9 (18 samples). When examining cluster 11(the right branch of observation 602 node), we find no samples of outcome 0, only 3 occurrences of outcome 2, which is not a majority value to conclude this feature is responsible for these outcomes. At the same time, it is also unclear whether this feature is important for outcome 7, given that it consists of 38 samples, accounting for approximately half of the subset. Similarly, outcome 9 is represented by only 1 observation.

In summary, the feature *crypto_Cipher_getinstance* does not exhibit the necessary qualities to distinctly separate certain outcomes from other families.
Now, shifting our focus to the 411 observations node, the relevant feature is *READ_SMS*. Cluster 11 comprises only 140 of the outcome 3 samples, compared to the 342 samples at the observations node. Thus, *READ_SMS* is not a crucial feature for distinguishing outcome 3 from other outcomes.
In conclusion, in this subsection, we have conducted a parameter tuning process and obtained the best parameter combination, specifically settling the key parameter *max_leaves* at a value of 8. Furthermore, we have focused on extracting the small branches from the generated results to provide insights into identifying key features that will be presented in the interpretable table in the next chapter. To obtain a more comprehensive quantitative evaluation of the entire tree, it will be further examined through the interpretable result table, which also will be introduced in the upcoming chapter.

5.4 Comparison Result

Table 5.3: Clustering accuracy comparison for each top 10 families among algorithm `CART`, `ExKMC` and `ICOT`. Each column represents an Andriod malware family, the last column is the average clustering accuracy for the algorithm. The symbol of "/" denotes no result for this family under this algorithm, and the bold numbers indicates the best results in these three algorithms.

	Airpush	Banbot	Dowgin	DroidKungFu	FakeInst	Fusob	Jisut	Kuguo	Mecor	Youmi	Average
`CART`	0.7036	0.6955	/	0.3972	0.7845	**0.4319**	/	0.5275	**0.7385**	0.5128	0.4792
`ExKMC`	0.7436	0.4945	/	0.5284	0.7368	0.4194	/	0.7334	0.3590	1	0.6269
`ICOT`	**0.8974**	**0.8352**	**0.6868**	**0.8993**	**0.7916**	0.2805	**0.4186**	**1**	0.6996	0.8462	**0.7355**

After tunning the parameters for each algorithm, we obtain the clustering accuracy result for each family in Table 5.3. We observe that algorithm `ICOT` gets the best result on the average clustering with 0.7355, and `ICOT` outperforms compared other two algorithms in 8 families, and `CART` has a better result on family Fusob and family Mecor. Moreover, on the average accuracy, `ExKMC` has an improvement of 0.1477. Therefore, algorithm `ICOT` has the best performance, and in the next chapter, we will use the interpretation result from algorithm output to better explain the logic behind algorithms on Android malware family clustering.

Chapter 6

Interpretation Result

In this chapter, our primary objective is to validate the efficacy of the ExKMC algorithm, while also conducting a comparative analysis between ExKMC and two other algorithms, namely ICOT and baseline CART. The aim is to examine their capabilities in explaining the clustering family of Android malware.

6.1 ExKMC Interpretable Result

In this section, the interpretable result of ExKMC will be our central focus of observation. We will draw conclusions based on the generated explanation tree and extract key features to display clustering results for each family. Subsequently, we will evaluate the generated results against the true labels.

Ground Truth Sources. When comparing the generated results with true labels, we have utilized a variety of reliable sources for establishing the ground truth. The dataset provided by AMD [116] explicitly mentions the references used for extracting the ground truth features, which include reputable third-party sources such as Dr.Web virus [1], Symantex[2] and etc. These sources have been analyzed by experts to ensure accuracy. Additionally, instead of relying solely on outdated datasets, we have validated the key features by consulting well-known antivirus comparisons in the current industry. By combining these diverse sources of information, we have extracted the ground truth for each malware family. This comprehensive approach allows us to effectively evaluate and compare the results obtained from the key features and the established ground truth.

Semantic Meaning. Since the results generated from algorithms have been condensed to only include extracted programming features, it is crucial to enhance our understanding of these key features in terms of their semantic meaning. For example, the key feature

[1] https://vms.drweb.com/virus/?i=4059936&lng=en

[2] https://www.broadcom.com/support/security-center

Url;OpenConnection may have a connection to the topic, but to fully comprehend how malware can exploit this feature and its impact on the device, we need to uncover the true significance of *Url;OpenConnection*. Consequently, we cross-reference each key feature with the Android documentation [1] to determine its meaning. According to the documentation, *Url;OpenConnection* refers to establishing a connection to a specific URL and engaging with the resource located at that URL. Armed with this understanding, we can proceed with our matching process accordingly.

Interpretation Result. The results of the ExKMC algorithm are presented in Table 6.1. The first column of the table represents the names of the malware families, and the serial numbers indicate the cluster numbers assigned by the algorithm. For example, the DroidKungFu family has the highest similarity value for both cluster 5 and cluster 8, so we mark them both for this family in the first column. In addition, the ground truth also includes masked serial numbers, indicating that they correspond to specific data points from the semantic meaning (the third column).

Evaluation Metrics. The F1-score is a commonly used evaluation measure to quantify the results obtained from a classifier [84]. However here, the task is not a classification but rather a clustering and also the evaluation for comparisons for the result of retrieved and the ground truth, therefore, we borrow the concept of F1-score in information retrieval.

The F1-score [109] combines precision and recall. Precision represents the fraction of relevant instances among the retrieved instances, while recall refers to the fraction of relevant instances that are retrieved in the model's output. The F1-score computes the harmonic mean of these two values, returning a ratio rather than the traditional arithmetic mean. A higher value of 1 indicates better performance for the F1-score. The equation of F1-score is as follows:

$$F = 2 * \frac{Precision * Recall}{Precision + recall}$$

Ground Truth Concepts. The features generated from the ExKMC algorithm and the ground truth are both in an unstructured context. Therefore, when comparing and evaluating them, it is necessary to consider individual words or phrases instead of the overall meaning [119].

For instance, the ground truth for the feature "Retrieve device info such as GPS and phone number" actually consists of three distinct concepts: retrieving device information, collecting GPS location information, and collecting the phone number. When computing evaluation metrics, such as recall, if we only have the generated features "Retrieve location and location updates", the total number of relevant features would be three instead of one in this case. To provide clarity, we have included the concepts for all 10 malware families in Table 6.2, which offers a detailed overview of the concepts associated with each family.

Ground Truth Example. Taking Airpush as an example, we refer to Table 6.3 to

	Key Features	Semantics Matching [1]	Ground Truth [116]
4. Airpush	Url;OpenConnection	1. Establish a connection to a URL and interact with the resource located at that URL.	A. Pushes advertisement from the Airpush network to the device. (1) B. Display content on the device notification bar. (8) C. Retrieve device info such as GPS, and phone number. (2.3.4.9)
	Context;getSystemService	2. Access to system resources and obtain the device information or policies.	
	requestLocationUpdates	3. Retrieve the device's current location and monitor location updates.	
	getBestProvider	4. Retrieve information about a specific location provider available on the device.	
	getLastKnownlocation	5. Retrieve the last known location of the device.	
	Cipher;init	6. Initialize the cipher with a specific mode.	
	getPackageInfo	7. Retrieve overall information about an installed package.	
	NotificationManageer;notify	8. Display a notification to the user.	
	getAccounts	9. Retrieve the user's accounts such as email accounts, social media accounts, phone numbers, or others.	
1. BankBot	context;getSystemService	1. Access to system resources and obtain the device information or policies.	A. Steals personal information(accounts, phone number, etc). (1) B. Send stolen information to a remote internet server. (2,8,9) C. Monitor SMS and send SMS messages.
	URLConnection;connect	2. Initiates the network connection and allows you to communicate with the remote server.	
	Acticity;getSystemService	3. Retrieve system-level services or managers within an activity.	
	Cipher;init	4. Initialize the cipher with a specific mode.	
	MediaPlayer;stop	5. Stop the playback of audio or video.	
	Throwable;printStackTree	6. Print the stack trace of an exception or error to the standard error stream.	
	getBestProvider	7. Determine the best available location provider based on the specified criteria.	
	URL;openConnection	8. Establish a connection to the resource specified by a URL.	
	getAllNetworkInfo	9. Retrieve the network connectivity state of all available network interfaces.	
	TelephonyManager;getCellLocation	10. Retrieve the current location of the device in terms of the cellular network.	
5.8 DroidKungFu	context;getSystemService	1. Access to system resources and obtain the device information or policies.	A. Collect device information, network information, and phone data. (1,4) B. Send collected information to remote servers. (2) C. Exploit vulnerabilities to root the device. (3) D. May install other applications onto the device.
	SMSManager;sendMultipartTextMessage (5)	2. Send an SMS message that exceeds the character limit of a single SMS.	
	Settings.System.putInt (5)	3. Modify system settings stored in Settings.System class.	
	getDeviceId (8)	4. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	
7. FakeInst	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Send the SMS. B. Receive commands from a remote server, (2)
	Url;OpenConnection	2. Establish a connection to a URL and interact with the resource located at that URL.	
	getDeviceId	3. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	
	Runtime;exec	4. Execute a command in the underlying operating system.	
0. Fusob	runtime;exec	1. Run external commands or execute shell scripts from within your Android application.	A. Sending/receiving SMSs. B. Lock phones by overlaying and displaying fake screens. (2) C. Steal network information such as Wi-Fi connection details. (3) D. Communicate to the remote server controlling the ransomware attack. (5)
	PowerManager$WakeLock;acquire	2. Acquire a wake lock (make the device awake all the time for displaying).	
	getActiveNetworkInfo	3. Obtain information about the currently active network connection.	
	getDeviceId	4. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	
	URL;openConnection	5. Establish a connection to the resource specified by a URL.	
2. Kuguo	context;getSystemService	1. Access to system resources and obtain the device information or policies.	A. Uses special library to hide executable bytecode. (2,4) B. Gets location, network, phone status(number,IMEI,etc) information. (3,5,8) C. Displays its own windows (ads) over windows of other apps.
	System;putInt	2. Modify system settings stored in Settings.System class.	
	URL;openConnection	3. Establish a connection to a URL and interact with the resource located at that URL.	
	getPackageInfo	4. Retrieve information about an installed package.	
	getAccounts	5. Retrieve the user's accounts such as email accounts, social media accounts, phone numbers, or others.	
	getAllvisitedUrls	6. Retrieve all visited URLs from the browser history.	
	MediaPlayer;reset	7. Release any resources associated with MediaPlayer object and prepare it for subsequent use.	
	getActiveNetworkInfo	8. Obtain information about the currently active network connection.	
3. Mecor	context;getSystemService	1. Access to system resources and obtain the device information or policies.	A. Collect device info, phone number, GPS, (1,5) B. Send stole information to a remote server. (4)
	Activity;getSystemService	2. Retrieve system-level services or managers within an activity.	
	Cipher;init	3. Initialize the cipher with a specific mode.	
	URL;openConnection	4. Establish a connection to a URL and interact with the resource located at that URL.	
	getLastKnownLocation	5. Retrieve the last known location of the device.	
6.9 Youmi	PowerManager$WakeLock;release (6)	1. Release a previously acquired wake lock (return to normal power-saving behavior).	A. Collect personal information and user contacts from the device. (8 B. Access the device's location. C. Display pop-up advertisements on the screen. (1,2) D. Display notifications and warnings. E. Receive commands from a remote server. (3)
	PowerManager$WakeLock;acquire (6)	2. Acquire a wake lock (make the device awake all the time for displaying).	
	getActiveNetworkInfo (9)	3. Obtain information about the currently active network connection.	
	url;openConnection(9)	4. Establish a connection to a URL and interact with the resource located at that URL.	
	MediaPlayer;release(9)	5. Release the MediaPlayer resources when you are done using it.	
	getAllNetworkInfo (9)	6. Retrieve the network connectivity state of all available network interfaces.	
	setVideoPath (9)	7. Set the path or URL of a video file to be played.	
	context;getSystemService (9)	8. Access to system resources and obtain the device information or policies.	
	System;putInt(9)	9. Modify system settings stored in Settings.System class.	

Table 6.1: Interpretation result on `ExKMC` algorithm

Table 6.2: Table of Concepts for Each Ground Truth of the Top 10 Families in the AMD Dataset. These concepts are necessary because the ground truth obtained from reliable resources is often in unstructured text, and a single sentence could possess multiple meanings, leading to difficulties in comparing the semantic meaning to the ground truth one by one. Therefore, initially, we extract the concepts from the ground truth and subsequently compare the semantic results with extracted ground truth concepts for evaluation. Therefore, when calculating the recall based on the ground truth, the denominator is the number of concepts for each ground truth. Additionally, to ensure consistency with the original resources on ground truth, we retain the original ground truth in the tables of interpretation results for each algorithm.

	Concepts from Ground Truth [116]
Airpush	a. From the network to the device. b. Notification bar. c. Retrieve device info. d. GPS. e. Phone number.
Banbot	a. Steal personal info. b. Send to a remote server. c. Send SMS.
Dowgin	a. Notification. b. Download applications. c. Collect device information. d. Send to a remote server.
DroidKungFu	a. Retrieve device information. b. Collect network information. c. Collect phone data. d. Send to a remote server. e. Boot the device. f. Install applications.
FakeInst	a. Send SMS. b. Receive commands from the server.
Fusob	a. Send and receive SMS. b. Lock phones. c. Steal network information. d. Communicate to the remote server.
Jisut	a. Display ads on the device (override keys, get the boot, kill the process). b. Send to a remote server.
Kuguo	a. Hide executable bytecode. b. Get the location. c. Steal network information. d. Steal device information. e. Display ads on the device.
Mecor	a. Steal personal info. b. Phone number. c. GPS. d. Send to a remote server.
Youmi	a. Collect personal information. b. Access the device's location. c. Display ads on the screen. d. Notifications. e. Receive from remote.

establish the correspondence between the generated feature meanings and the ground truth. The precision is calculated as the ratio between the correct labels and the returned features, which in this case is 5/9. The recall represents the fraction of correct items among all relevant labels, yielding a value of 5/5. Furthermore, the F1-score is computed as the harmonic mean of precision and recall, resulting in a value of 0.7143. Based on these measurements, we can now determine the precision and recall values for each family in the generated `ExKMC` result, following the aforementioned rule.

Table 6.3: Table of concepts of each ground truth for family **Airpush**

	Concepts
Ground truth [116]	A. Network to the device (1) B. Notification bar (8) C. Retrieve device info (2) D. GPS (3,5) E. Phone number (9)
Generated features meaning	1. Establish a connection to fetch resources. 2. Obtain the device information. 3. Retrieve location and location updates. 4. Retrieve the location provider. 5. Retrieve the last-known location. 6. Initialize the cipher. 7. Retrieve info about the installed package. 8. Display a notification. 9. Obtain info like phone numbers and accounts.

Evaluation Result. Table 6.4 presents the precision, recall, and F1-score results obtained from the `ExKMC` algorithm. For the Dowgin and Jisut families, the algorithm `ExKMC` fails to detect them, as indicated by the "/" symbol in the table, the same rule is also applied to the subsequent tables and figures.

From a precision perspective, it measures the accuracy of positive predictions generated by the algorithm. The DroidKungFu family achieves the best results. By examining Table 6.1, we can observe that compared to the 9 features generated by Airpush and the 10 features generated by Bankbot. Family DroidKungFu only has 4 predicted key features, indicating a higher potential for precision, since family DroidKungFu has only 6 ground truth labels. Furthermore, three of four predicted key features correctly identify the ground truth labels, leading to higher precision.

Regarding recall, it measures the model's ability to correctly identify all positive instances. In terms of recall values, Airpush exhibits the best results. Although it may be deficient in precision due to generating a large number of key features, it successfully identifies all three ground truths. Therefore, for the F1-score, the Airpush family achieves the highest result of 0.7143, mainly due to its high recall value.

In conclusion, by setting a separation threshold of 0.3, all top 10 families on AMD dataset can attain this level of F1-score for the interpretation result. Specifically, the highest results are achieved by the Airpush and DroidKungFu families, both surpassing 0.7. Conversely, the lowest results are observed for the undetected families Dowgin and Jisut.

Table 6.4: Precision, Recall, and F1-score from ExKMC results

	Airpush	Banbot	Dowgin	DroidKungFu	FakeInst	Fusob	Jisut	Kuguo	Mecor	Youmi	Average
Precision	0.5556	0.2	/	**0.6667**	0.25	0.6	/	0.5	0.6	0.3333	0.3706
Recall	**1.0**	0.6667	/	0.75	0.5	0.75	/	0.8	0.75	0.6	0.5817
F1-score	**0.7143**	0.3077	/	0.7059	0.3333	0.6667	/	0.6154	0.6667	0.4286	0.4439

6.2 Comparison Interpretation Result on ExKMC, ICOT and CART

In this section, first, we will present two other comprehensive interpretation result tables for algorithm ICOT and CART. Next, we will select three representative families (Airpush, DriodKungFu, and Youmi) and examine the features generated from these algorithms. This analysis will allow us to delve into a more detailed comparison of results at the family level. Finally, the evaluation results, measured by the F1-score, will be displayed in order to provide an overview of the quality value for the Interpretation Result on algorithm ExKMC, ICOT and CART.

For the ICOT algorithm, we will use the results obtained with a maximum of 6 leaves as the final parameter. Since ICOT shares the same foundation as ExKMC, both algorithms are based on the k-Means method and employ the k-Means labels to facilitate interpretability. Therefore, it is feasible to control CART and ExKMC using the same parameter. The only parameter to adjust is the maximum number of leaves, which is set as $8 * k$, where $k = 10$.

Interpretation Table for Algorithms ICOT and CART. For the results, Table 6.5 presents the interpretation results obtained from the ICOT algorithm. The CART interpretation results can be found in Table 6.6.

Table 6.5: Interpretation result on ICOT algorithm

	Key Features	Semantics Matching [1]	Ground Truth [116]
0. Airpush	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Pushes advertisement from the Aripush network to the device. B. Display content on the device notification bar. C. Retrieve device info such as GPS, and phone number. (1,5)
	getActiveNetworkInfo	2. Obtain information about the currently active network connection.	
	MEDIA_EJECT	3. Remove the external storage media.	
	BOOT_COMPLETED	4. A broadcast intent when the device completes the booting process and becomes fully operational.	
	READ_PHONE_STATE	5. Access to phone state (current network information, any ongoing calls status, and phone Accounts).	
	category.HOME	6. Represents the main interface when they unlock their device or press the home button.	
1. BankBot	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Steals personal information(accounts, phone number, etc). (1,4) B. Send stolen information to a remote internet server. C. Monitor SMS and send SMS messages. (1,2,3)
	Cipher_getParameters	2. Retrieve the algorithm parameters about the current encryption or decryption operation.	
	SEND_SMS	3. Allows an application to send SMS messages.	
	CALL_PHONE	4. Initiate phone calls directly without user intervention.	
	category.HOME	5. Represents the main interface when they unlock their device or press the home button.	
2. Dowgin	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Display advertisements in the system notification bar. B. Download and request installation of new applications. C. Send device information and details about installed applications to a remote location. (1,2,3,4,5)
	getActiveNetworkInfo	2. Obtain information about the currently active network connection.	
	READ_PHONE_STATE	3. Access to phone state (current network information, any ongoing calls status, and phone Accounts).	
	ACCESS_FINE_LOCATION	4. Access the device's GPS or network-based location data with higher accuracy.	
	ACCESS_COARSE_LOCATION	5. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.	
3. DroidKungFu	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Collect device information, network information, and phone data. (1,2,5) B. Send collected information to remote servers. C. Exploit vulnerabilities to root the device. (3,4) D. May install other applications onto the device.
	getActiveNetworkInfo	2. Obtain information about the currently active network connection.	
	BOOT_COMPLETED	3. A broadcast intent when the device completes the booting process and becomes fully operational.	
	Runtime;exec	4. Run external commands or execute shell scripts from within your Android application.	
	READ_PHONE_STATE	5. Access to phone state (current network information, any ongoing calls status, and phone Accounts).	
	ACCESS_COARSE_LOCATION	6. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.	
4. FakeInst	SEND_SMS	1. Allows an application to send SMS messages.	A. Send the SMS. (1) B. Receive commands from a remote server.
	getLine1Number	2. Retrieve the phone number associated with the SIM card in the device.	
5. Fusob	BOOT_COMPLETED	1. A broadcast intent when the device completes the booting process and becomes fully operational.	A. Send or receive SMSs. B. Lock phones by overlaying and displaying fake screens. C. Steal network information such as Wi-Fi connection details. D. Communicate to the remote server controlling the ransomware attack.
	READ_CALL_LOG	2. Read the call history or call log of the device.	
6. Jisut	getActiveNetworkInfo	1. Obtain information about the currently active network connection.	A. Display advertising content in the process(override keys, get boot, kill process). (2,3) B. Gather and forward details information from the device.
	BOOT_COMPLETED	2. A broadcast intent when the device completes the booting process and becomes fully operational.	
	Runtime;exec	3. Run external commands or execute shell scripts from within your Android application.	
7. Kuguo	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Uses special library to hide executable bytecode. B. Gets location, network, phone status (number,IMEI,etc) information. (1,2,3,4) C. Displays its own windows (ads) over windows of other apps.
	getActiveNetworkInfo	2. Obtain information about the currently active network connection.	
	READ_PHONE_STATE	3. Access to phone state (current network information, any ongoing calls status, and phone Accounts).	
	ACCESS_COARSE_LOCATION	4. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.	
8. Mecor	getAccounts	1. Retrieve the user's accounts such as email accounts, social media accounts, phone numbers, or others.	A. Collect device info, phone number, GPS. (2/3) B. Send stole information to a remote server.
9. Youmi	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Collect personal information and user contacts from the device. (1,3) B. Access the device's location. (4) C. Display pop-up advertisements on the screen. D. Display notifications and warnings. E. Receive commands from a remote server.
	Runtime;exec	2. Run external commands or execute shell scripts from within your Android application.	
	READ_PHONE_STATE	3. Access to phone state (current network information, any ongoing calls status, and phone Accounts).	
	ACCESS_COARSE_LOCATION	4. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.	

6.2. Comparison Interpretation Result on ExKMC, ICOT and CART

Table 6.6: Interpretation result on CART algorithm

	Key Features	Semantics Matching [1]	Ground Truth [116]
4. Airpush	getActiveNetworkInfo	1. Obtain information about the currently active network connection.	
	getSystemService	2. Access to system resources and obtain the device information or policies.	A. Pushes advertisement from the Aripush network to the device.
	URL;openConnection	3. Establish a connection to a URL and interact with the resource located at that URL.	B. Display to the device notification bar.
	URLConnection;connect	4. Initiates the network connection and allows you to communicate with the remote server.	C. Retrieve device info such as GPS, phone number. (2,5)
	getBestProvider	5. Determine the best available location provider based on the specified criteria.	
1. BankBot	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Steals personal information(accounts, phone number, etc). (1,2)
	getDeviceId	2. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	B. Send stolen information to a remote internet server.
	MediaPlayer;start	3. Start or resume playback of an audio or video file.	C. Monitor SMS and send SMS messages.
5.8. DroidKungFu	URL;openConnection	1. Establish a connection to a URL and interact with the resource located at that URL.	A. Collect device information, network information, phone data. (5,7)
	Cipher;init	2. Initialize the cipher with a specific mode.	B. Send collected information to remote servers. (1.3.4)
	URLConnection;connect(8)	3. Initiates the network connection and allows you to communicate with the remote server.	C. Exploit vulnerabilities to root the device. (2)
	getActiveNetworkInfo	4. Obtain information about the currently active network connection.	D. May install other applications onto the device.
	getPackageInfo(8)	5. Retrieve overall information about an installed package.	
	getDeviceId	6. Control the vibrating motor (also known as a vibrator) on the device.	
	Vibrator;vibrate(8)	7. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	
	getAllNetworkInfo(8)		
7. FakeInst	getDeviceId	1. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	A. Send the SMS.
	URL;openConnection	2. Establish a connection to a URL and interact with the resource located at that URL.	B. Receive commands from a remote server. (2)
	getSystemService	3. Access to system resources and obtain the device information or policies.	
0. Fusob	URL;openConnection	1. Establish a connection to a URL and interact with the resource located at that URL.	A. Sending/receiving SMSs.
	getDeviceId	2. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	B. Lock phones by overlaying and displaying fake screens.
			C. Steal network information such as Wi-Fi connection details.
			D. Communicate to the remote server controlling the ransomware attack. (1)
2. Kuguo	Runtime;exec	1. Run external commands or execute shell scripts from within your Android application.	A. Uses special library to hide executable bytecode.
	getActiveNetworkInfo	2. Obtain information about the currently active network connection.	B. Gets location, network, phone status(number, IMEI, etc) information. (1/3)
	URL;openConnection	3. Establish a connection to a URL and interact with the resource located at that URL.	C. Displays its own windows (ads) over windows of other apps.
3. Mecor	getSystemService	1. Access to system resources and obtain the device information or policies.	A. Collect device info, phone number, GPS. (1/3)
			B. Send stole information to a remote server.
6.9. Youmi	getActiveNetworkInfo(6)	1. Obtain information about the currently active network connection.	
	NotificationManager;notify(6)	2. Display a notification to the user.	A. Collect personal information and user contacts from the device. (6)
	MediaPlyaer;release(6)	3. Release the MediaPlayer resources when you are done using it.	B. Access the device's location.
	getBestProvider(9)	4. Determine the best available location provider based on the specified criteria.	C. Display pop-up advertisements on the screen. (3,7)
	getPackageInfo(9)	5. Retrieve overall information about an installed package.	D. Display notifications and warnings. (2)
	getDeviceId(9)	6. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.	E. Receive commands from a remote server.
	MediaPlayer;stop(9)	7. Stop the playback of audio or video files.	

Table 6.7: This table shows the interpretation results of three algorithms. Each column represents the result of an individual algorithm. For each family, there are three components: key features generated by the algorithm, their corresponding semantic meanings, and the ground truth for comparison with the semantic meanings. Corrected identified features will be marked with an asterisk (*).

	ExKMC	ICDT	CART
Airpush	*Key Features* Url;OpenConnection Context;getSystemService requestLocationUpdates getBestProvider getLastKnownlocation Cipher;init getPackageInfo NotificationManageer;notify getAccounts	*Key Features* getSystemService getActiveNetworkInfo MEDIA_EJECT BOOT_COMPLETED READ_PHONE_STATE category.HOME	*Key Features* getActiveNetworkInfo getSystemService URL;openConnection URLConnection;connect getBestProvider
	Semantic Meaning [1] 1. Establish a connection to a URL and interact with the resource located at that URL. * 2. Access to system resources and obtain the device information or policies. * 3. Retrieve the device's current location and monitor location updates. * 4. Retrieve information about a specific location provider available on the device. * 5. Retrieve the last known location of the device. 6. Initialize the cipher with a specific mode. 7. Retrieve overall information about an installed package. 8. Display a notification to the user. 9. Retrieve the user's accounts such as email accounts, social media accounts, phone numbers, or others. *	*Semantic Meaning* [1] 1. Access to system resources and obtain the device information or policies. * 2. Obtain information about the currently active network connection. 3. Remove the external storage media. 4. A broadcast intent when the device completes the booting process and becomes fully operational. 5. Access to phone state (current network information, any ongoing calls status, and phone Accounts). * 6. Represents the main interface when they unlock their device or press the home button.	*Semantic Meaning* [1] 1. Obtain information about the currently active network connection. 2. Access to system resources and obtain the device information or policies. * 3. Establish a connection to a URL and interact with the resource located at that URL. 4. Initiates the network connection and allows you to communicate with the remote server. 5. Determine the best available location provider based on the specified criteria. *
	Ground Truth [116] :1. Pushes advertisement from the Aripush network to the device. 2. Display content on the device notification bar. 3. Retrieve device info such as GPS, and phone number.		
DroidKungFu	*Key Features* context;getSystemService SMSManager;sendMultipartTextMessage Settings.System.putInt getDeviceId	*Key Features* getSystemService getActiveNetworkInfo BOOT_COMPLETED Runtime;exec READ_PHONE_STATE ACCESS_COARSE_LOCATION	*Key Features* URL;openConnection Cipher;init URLConnection;connect getActiveNetworkInfo getPackageInfo getDeviceId Vibrator;vibrate getAllNetworkInfo
	Semantic Meaning [1] 1. Access to system resources and obtain the device information or policies.* 2. Send an SMS message that exceeds the character limit of a single SMS.* 3. Modify system settings stored in Settings.System class. 4. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.*	*Semantic Meaning* [1] 1. Access to system resources and obtain the device information or policies.* 2. Obtain information about the currently active network connection.* 3. A broadcast intent when the device completes the booting process and becomes fully operational. * 4. Run external commands or execute shell scripts from within your Android application.* 5. Access to phone state (current network information, any ongoing calls status, and phone Accounts).* 6. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.*	*Semantic Meaning* [1] 1. Establish a connection to a URL and interact with the resource located at that URL.* 2. Initialize the cipher with a specific mode. 3. Initiates the network connection and allows you to communicate with the remote server.* 4. Obtain information about the currently active network connection.* 5. Retrieve overall information about an installed package. 6. Control the vibrating motor (also known as a vibrator) on the device. 7. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.*
	Ground Truth [116] : 1. Collect device information, network information, and phone data. 2. Send collected information to remote servers.3. Exploit vulnerabilities to root the device.4. May install other applications onto the device.		
Youmi	*Key Features* PowerManager$WakeLock;release (6) PowerManager$WakeLock;acquire (6) getActiveNetworkInfo (9) url;openConnection(9) MediaPlayer;release(9) getAllNetworkInfo (9) setVideoPath (9) context;getSystemService (9) System;putInt(9)	*Key Features* getSystemService Runtime;exec READ_PHONE_STATE ACCESS_COARSE_LOCATION	*Key Features* getActiveNetworkInfo(6) NotificationManager;notify(6) MediaPlyaer;release(6) getBestProvider(9) getPackageInfo(9) getDeviceId(9) MediaPlayer;stop(9)
	Semantic Meaning [1] 1. Release a previously acquired wake lock (return to normal power-saving behavior).* 2. Acquire a wake lock (make the device awake all the time for displaying).* 3. Obtain information about the currently active network connection.* 4. Establish a connection to a URL and interact with the resource located at that URL.* 5. Release the MediaPlayer resources when you are done using it. 6. Retrieve the network connectivity state of all available network interfaces. 7. Set the path or URL of a video file to be played. 8. Access to system resources and obtain the device information or policies.* 9. Modify system settings stored in Settings.System class.	*Semantic Meaning* [1] 1. Access to system resources and obtain the device information or policies.* 2. Run external commands or execute shell scripts from within your Android application. 3. Access to phone state (current network information, any ongoing calls status, and phone Accounts).* 4. Retrieve location data with lower accuracy such as cell tower triangulation or Wi-Fi positioning.*	*Semantic Meaning* [1] 1. Obtain information about the currently active network connection.* 2. Display a notification to the user.* 3. Release the MediaPlayer resources when you are done using it. 4. Determine the best available location provider based on the specified criteria.* 5. Retrieve overall information about an installed package. 6. Retrieve the unique device identifier (IMEI, MEID, or ESN) of the phone.* 7. Stop the playback of audio or video files.
	Ground Truth [116] :1. Collect personal information and user contacts from the device. 2. Access the device's location. 3. Display pop-up advertisements on the screen. 4. Display notifications and warnings. 5. Receive commands from a remote server.		

6.2. Comparison Interpretation Result on ExKMC, ICOT and CART

Interpretation Result Comparison. We then compare the interpretation result on three algorithms, and demonstrate three representative malware families (Airpush, DriodKungFu, and Youmi), as shown in Table 6.7. Figure 6.1 describes the average F1-score result for all malware families for algorithm ExKMC, ICOT, and CART. And the detailed analysis and family study are listed as follows:

- **Airpush**
  The Airpush family is chosen as the representative for this research since it has the largest number of ten families. The ground truth for Airpush involves pushing advertisements from the Airpush network to the device. Both ExKMC and CART detect it through the feature *openConnection*. Additionally, ExKMC also detects the display of content on the device notification bar. All three algorithms (ExKMC, ICOT, and CART) detect the retrieval of device information using the method *getSystemService*. For the ground truth of GPS usage, both ExKMC and CART achieve accurate detection. Moreover, ExKMC successfully detects the phone number through *getAccounts*, while ICOT achieves this by reading the phone state.

- **Bankbot**
  For other noticeable families, such as the Bankbot family, it achieves the lowest F1-score in Figure 6.1, with all scores hovering around 0.3. In the case of ExKMC, it extracts a large number of features, but only 4 of them are relevant, and 2 out of those 4 features have the same meaning, namely, collecting information from *getSystemService*. In comparison, ICOT extracts 5 features, which is half the number extracted by ExKMC, and 2 out of those 5 features are correctly identified according to the precision calculation. As for the results from CART, it also extracts a limited number of features, but it only detects 1 out of the 3 important features. Therefore, the ICOT algorithm performs better in this context.

- **Dowgin**
  Upon observing the similarity table and examining the distribution of the Dowgin family, it becomes evident that it exhibits a similarity of approximately 0.1 within each cluster. This raises the question of whether the features of Dowgin are not distinct enough to differentiate it from other families. In the case of Dowgin, the ground truth feature of displaying advertisements on the screen is present in only 1 out of 2 ground truth families, while the feature of downloading and installing new applications is found in only 2 out of 10 families. However, 9 out of 10 families share the feature of connecting to remote servers. The failure to distinguish Dowgin from others may stem from the shared features it has with other families and the lack of common features related to installing applications. The *in-*

tent.ACTION_INSTALL_PACKAGE feature can be considered in this context, but unfortunately, the algorithm does not detect it.

- **DroidKungFu**
  The F1-score for both families exhibits minimal variance, with `ICOT` yielding a higher score of approximately 0.8333, followed by `ExKMC` at around 0.7, and `CART` performing the poorest with a score of 0.6154. Both `ExKMC` and `ICOT` extract the *getSystemService* feature, which indicates their ability to identify the relative features associated with stealing device information, a characteristic of the DroidKungFu ground truth. `ICOT` and `CART` both extract relevant information related to obtaining internet information through the *getActiveNetworkInfo* feature. Furthermore, all three algorithms extract features related to collecting phone data, (*getDeviceId*, *read_phone_state*, and *getDeviceId*). Regarding the ground truth of sending information to remote servers, `ExKMC` and `CART` both achieve this through features like *sendMultipleTextMessage* and *URL.openConnection*. One notable distinguishing feature of `ICOT` is its ability to identify the ground truth feature of exploiting vulnerabilities to boot the device, detected through *boot_completed* and *Runtime.exec*. However, none of the three algorithms detect the ground truth probability of installing applications onto the device. Given that this ground truth contains the verb "may", it is likely that the number of malware samples presenting this feature is relatively small, resulting in the inability of these three technologies to detect it accurately.

- **FakeInst**
  For the FakeInst family, a confusion matrix score of 1 in Figure 4.4 has shown a sign of excellent performance. Among the interpretation results, the `ICOT` algorithm achieves the highest F1-score of approximately 0.5, while `CART` and `ExKMC` show similar performance. For family FakeInst, there are only two clearly identifiable true labels. Among the four generated results from `ExKMC`, it correctly identifies the second true label. For the `ICOT` algorithm, it generates only two key features, and only one of them correctly identifies the true label related to sending SMS messages, while overlooking the crucial feature related to network behavior. For `CART` algorithm, similarly to the `ExKMC` results, it also identifies only one feature correctly, specifically regarding SMS messages.

- **Fusob & Jisut**
  However, the reason for the confusion in detecting the Jisut family is different from that of Dowgin. According to the similarity table (Table 5.1), Jisut has the highest similarity with cluster 0. However, cluster 0 shows the highest similarity with the Fusob family. This indicates that there is a significant similarity between Fusob and

Jisut. Consequently, it is necessary to thoroughly examine the ground truth of the Fusob and Jisut families in order to gain a better understanding of the similarities and differences between them.

In the ground truth of the Fusob family, it is identified that they engage in the following activities: 1. Sending or receiving SMS messages. 2. Locking phones by overlaying and displaying fake screens. 3. Stealing network information such as Wi-Fi connection details. 4. Communicating with the remote server controlling the ransomware attack. On the other hand, the ground truth of the Youmi family includes the following activities: 1. Displaying advertising content by overriding keys, obtaining boot access, and killing processes. 2. Gathering and forwarding detailed information from the device.

We can observe that both families share the characteristics of displaying advertisements on the screen and communicating with a remote server to send stolen information. The distinguishing factor is that the Fusob family also involves sending and receiving SMS messages and stealing internet information. This suggests that the additional features of the Fusob family may make it stand out more distinctly. Additionally, since the ground truth of Fusob actually "includes" the features of Jisut, it explains the difficulty in separating Fusob and Jisut through the clustering algorithm. This also clarifies why ExKMC missed identifying Jisut, while ICOT missed identifying Fusob.

- **Kuguo**
  For the Kuguo family, the true labels consist of three features. Among the key features generated by ExKMC, five of them are actually correctly identified, which corresponds to two-thirds of the overall features in the true labels. For the ICOT algorithm, all four generated key features correctly identify the crucial true label of obtaining personal information, resulting in high performance for the Kuguo family on algorithm ICOT. For the CART algorithm, it generates three key features, but only one of them provides the best identification for one true label according to the ground truth.

- **Mecor**
  In total, all three algorithms achieve great F1-score results, ranging from 0.4 to 0.7, which indicates a strong performance. This is further supported by the confusion matrix score of 1 in Figure 4.4 from Chapter 4. Additionally, for both true labels, ExKMC successfully identifies them using three out of the five generated features. For the ICOT algorithm, it only produces one feature related to retrieving personal information, which is specifically associated with the Mecor family. As for CART, it

generates a relatively general key feature pertaining to obtaining information and identifies only one true label.

- **Youmi**
  For the Youmi family, `CART` performs the best with an F-score of approximately 0.5, followed by `ICOT` with a score of 0.44, and `ExKMC` performs the worst with a score of 0.42. From the similarity table, Table 5.1, it is evident that Youmi exhibits similarities within each cluster. All three algorithms detect the feature of collecting personal information and user contacts from the device. Only `CART` and `ICOT` detect the attempt to access the device's location. Regarding the display of pop-up advertisements on the screen, `ExKMC` and `ICOT` extract the feature of displaying notifications and warnings, while `CART` detects this feature through the *notify* feature. Additionally, `ExKMC` and `CART` both detect the possibility of connecting to a remote server.

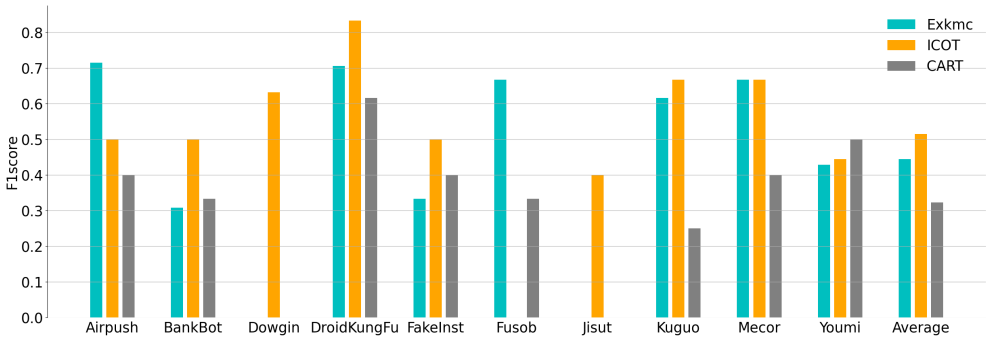


Figure 6.1: F1-score on algorithms of `ExKMC`, `ICOT` and `CART` interpretation result

Evaluation Result. Figure 6.1 presents the results for all malware families across the `ExKMC`, `ICOT`, and `CART` technologies. Through this figure, we can clearly observe which algorithm generates better results for information retrieval within each family. From the average F-score, it is evident that the `ICOT` algorithm outperforms `ExKMC`, which amounts to approximately 5% and both the up-to-date algorithms perform better than the baseline `CART` algorithm.

Upon closer examination of the table, we can observe that for the Dowgin and Jisut families, `ICOT` provides detailed explanations and relevant features. In contrast, both `ExKMC` and the baseline `CART` algorithm confuse Dowgin with DroidKungFu and Jisut with Youmi, resulting in both algorithms yielding a feature score of 0 for these families. Therefore, the results from `ICOT` are superior. Furthermore, Dowgin and Jisut achieve decent scores of 0.63 and 0.4, respectively. Although `ICOT` successfully predicts almost

ten families, it fails to correctly identify the Fusob family, as there is limited semantic meaning correspondence with the ground truth in Table 6.5.
Therefore, in total, as shown in Figure 6.1, ICOT exhibits the best performance with an F1 score exceeding 0.5. Additionally, among the ten families analyzed, ICOT outperforms other algorithms in seven of them, while ExKMC proves to be a decent alternative, particularly in the context of Airpush and Fusob. In terms of our own family, both algorithms demonstrate similar performance. However, when considering the differences, it becomes evident that CART fails due to its tendency to generate an excessive number of irrelevant features. Consequently, ExKMC can still deliver satisfactory results, although slightly inferior to ICOT, which was introduced one year later.

Chapter 7

Discussion

In this section, we will discuss the implications in Section 7.1 and limitations in Section7.2.

7.1 Implications

In this study, we investigate the performance of explanation unsupervised machine learning on Android malware applications using three algorithms: `ExKMC`, `ICOT`, and the baseline algorithm, `CART`. Next, we conducted experiments on a large-scale real-world malware dataset AMD. Based on the clustering accuracy, the algorithm `ICOT` outperformed `ExKMC` with a significantly better accuracy of 0.7355. `ExKMC` demonstrated the second-best performance with an accuracy of 0.6269. Both algorithms showed improvements compared to the accuracy of algorithm `CART`, which scored 0.4792. In the end, the interpretation results of key features present the underneath reasonable process of decision-making in machine learning. Among the algorithms, `ICOT` achieves the highest F1-score of 0.52.
In detail, based on the interpretation results for each family, we observe the key features through the generated explanation trees, which leads to further analysis or the detection of unfamiliar and ungrouped Android malware samples. Among these three algorithms, when comparing `ICOT` with the `ExKMC` algorithm, it is evident that algorithm `ExKMC` does not outperform `ICOT` on "accuracy". After examination of the interpretation results, it is apparent that for certain families (e.g. Bankbot and Youmi), `ExKMC` generates significantly more key features than `ICOT` algorithm (10 vs 5) and (9 vs 4) respectively, indicating a need for `ExKMC` to improve on accurate prediction. As for the `CART` algorithm, after conducting experiments and comparing it with `ExKMC`, we find that it generates a large number of irrelevant key features in most families, resulting in a final poor F1-score of 0.32.
For the dataset, the examination through large unlabeled datasets is crucial for experiments of algorithms, given the continuous emergence of malware families. The `ExKMC` and `ICOT` algorithms have shown great performance when tested on a large-scale real-world

malware dataset (AMD). In particular, `ExKMC` achieves an F1-score of 0.45 and `ICOT` achieves the highest F1-score of 0.52. In comparison to previous studies, such as Aung et al. [129] with 700 samples and Ali et al. [34] with 800 samples, which only tested with hundreds of samples, both `ExKMC` and `ICOT` algorithm demonstrate promising results for identifying malware families.

7.2 Limitations

In this thesis, we have proven the good performance of algorithm `ExKMC`, `ICOT`, and the baseline algorithm, `CART` on the application of clustering the Android malware family. However, it is important to acknowledge that our analysis has certain limitations during the process.

Assumption in k-Means. Before conducting experiments, we assume that we have studied 10 families with a known number of members to be discussed. This implies that we assume we know the value of k in advance. Also, in this thesis, determining the appropriate cluster for an unknown dataset is not within the scope of our research. However, in reality, there are cases where we do not know the number of families [34]. Therefore, accurately determining the exact number of target clusters requires careful experimentation. There are various methods available for this purpose, including but not limited to the Elbow Method [71] and the Silhouette Method [80]. Both methods can be further explored in future research.

Evaluation. When conducting the calculation in Chapter 6, there is some confusion about getting the precision result. According to traditional rules, precision is determined by dividing the number of correctly identified items by the total number of predictions. However, in the case where there are similar key features, such as *URL;openConnection* and *URLConnection;connect*, it is important to consider counting them as a single correct prediction to align with the ground truth. This is because there is only one ground truth related to URL connection. By not accounting for this consideration, the precision value may unfairly decrease, as it fails to capture the correct predictions. Therefore, it can be argued that incorporating this approach provides a better evaluation metric for precision.

Dataset. When conducting the experiments, we only tested on the real-world Android malware dataset (AMD), different datasets and more malware families are considered for testing the performance of the result.

Gap between Algorithm Result and Interpretation Result. There exists a gap between the evaluation results for the algorithms (e.g. `ExKMC` algorithm evaluation MoJoFM result, the clustering accuracy result for each family on each algorithm) and the final evaluation results from the interpretation table. This discrepancy raises the question

of whether the evaluation metrics can be compatible or consistent with the interpretation results. For instance, during the data preprocessing in Chapter 4, we attempted to scale the results, which is a common and essential practice in machine learning. Scaling ensures that the features or variables of a dataset are on a similar scale or range. From the results obtained, scaling the data in the `ExKMC` algorithm potentially increased the MoJoFM score from 67% to 85%. However, this scaling process does not necessarily improve the quality of the explanation table and F1-score. If we apply the scaled dataset to the interpretation result and compared them, obtaining the evaluation result via F1-score, we could observe the decrease from 0.44 to 0.11. Additional details regarding this matter can be found in Table 7.1. Therefore, it highlights the need for further research to explore the relationship between the algorithm's quantitative evaluation and interpretation result quality assessment.

Table 7.1: F1-score for the scaled datasets interpretation result on the `ExKMC` algorithm

Methods	Airpush	BankBot	Dowgin	DroidKungFu	FakeInstaller	Fusob	Jisut	Kuguo	Mecor	Youmi	mean
`ExKMC`	0.7143	0.3077	/	0.7059	0.3333	0.6667	/	0.6154	0.6667	0.4286	0.4439
`ExKMC_scalered`	0.0769	0.4000	/	0.2222	0	/	0	/	0.2222	0.2105	0.1132

Mapping between the Key Features and Ground Truth. When comparing the results of precision and recall, it is necessary to quantitatively determine whether they match. However, this process may involve subjectivity at times. Therefore, further research is required to either develop a more advanced NLP model that generates directly comparable results with the ground truth or conduct a survey to leverage the biases from a human perspective like Wu et al. [119].

For example, to avoid bias, an online survey could be conducted to gather opinions from a diverse range of participants. Participants should possess basic knowledge of malware analysis and preferably be experts in the field. Diversity among participants is important, encompassing various domains, countries, research institutions (including universities), and occupations such as PhD researchers, industry professionals, postdocs, or professors. Participants would be asked to rate the extent to which the generated semantic meaning truly matches the ground truth on a scale of 1 to 5, with 5 representing a perfect match and 1 indicating a poor match. Scaling the ratings in this manner allows for a more nuanced evaluation

Interpretaion Result. Currently, the interpretation results (as demonstrated by Table 6.5) reveal that the key features are the only output derived from the generated explanation tree by the algorithm (as illustrated in Figure 5.6). However, it's still unclear for

analysts to directly analyze the interpretation result in a more practical setting in the industry. Further research is required to delve into the semantic level of analysis to draw conclusions for unlabeled data. This would involve analyzing the interpretation results and, ultimately, developing strategies to mitigate malware attacks.

Chapter 8

Conclusions

In this thesis, we have studied two state-of-the-art approaches and one baseline model for unsupervised learning in applications to Android malware detection and family clustering. Here, it is important that the clustering approaches not only accurately classify malware samples into families, but also provide insights for security analysts why this family is chosen as the clustering target based on behavior-related features.
In our approach, we worked with the relatively recent Android malware dataset (AMD). We have experimented with feature selection and data preprocessing, and hyperparameter tuning of the three studied algorithms. We have then compared these algorithms on the task of clustering Android malware into 10 clusters. Our results show that algorithm `ICOT` is the best to achieve the highest clustering accuracy with 0.7355, and algorithm `ExKMC` also demonstrated an improvement over the baseline `CART` with an accuracy of 0.6269.
We then experimented with the capability of these algorithms to report the key features for the clustering decision and assessed them for being the basis for Android malware family identification, comparing to the ground truth with have assembled from reputable security sources and the Android documentation. Our results show that algorithm `ICOT` achieved the best F1-score of 0.52. As for algorithm `ExKMC`, it needs to improve its feature prediction accuracy further, as it generates more irrelevant features than algorithm `ICOT`. Our findings also demonstrate that there is a complex relationship between clustering accuracy and the interpretability of the results. This is an interesting observation that requires more attention from the field of malware research.

8.1 Future Work

As mentioned in Section 7.2, several research directions have emerged from this work. For the dataset, further examination is required to assess the effectiveness of algorithms on different malware families, since in this essay, we mainly focus on the study of one dataset.

Additionally, due to the selection process of malware families, it may be necessary to include a more comprehensive range of families. For example, testing the algorithms on the entire set of 71 families could provide valuable insights.
Regarding the algorithms themselves, it is imperative to notice that improvements can be made to `ExKMC` and `ICOT`. Furthermore, exploring other XAI unsupervised machine learning algorithms on Android malware families, such as neural networks, could be a solution to yield a better result.
For the evaluation of interpretation results, it may be beneficial to search for more effective and compatible metrics for comparing key features and true labels. To ensure accuracy and minimize bias, it is crucial to involve human experts in extracting true labels from websites and in the process of comparing the results between semantic meanings and true labels in the interpretation table.

Bibliography

[1] Android API reference[online]. `https://developer.android.com/reference`. 2023.

[2] Prerna Agrawal and Bhushan Trivedi. Machine learning classifiers for Android malware detection. In *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2020, Volume 1*, pages 311–322. Springer, 2021.

[3] Mohammed M Alani and Ali Ismail Awad. Paired: An explainable lightweight Android malware detection system. *IEEE Access*, 10:73214–73228, 2022.

[4] Waleed Ali. Hybrid intelligent Android malware detection using evolving support vector machine based on genetic algorithm and particle swarm optimization. *IJCSNS*, 19(9):15, 2019.

[5] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*, pages 468–471, 2016.

[6] Ebtesam J Alqahtani, Rachid Zagrouba, and Abdullah Almuhaideb. A survey on Android malware detection techniques using machine learning algorithms. In *2019 Sixth International Conference on Software Defined Systems (SDS)*, pages 110–117. IEEE, 2019.

[7] Marco Aresu, Davide Ariu, Mansour Ahmadi, Davide Maiorca, and Giorgio Giacinto. Clustering Android malware families by http traffic. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 128–135. IEEE, 2015.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of Android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[9] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion*, 58:82–115, 2020.

[10] Nida Aslam, Irfan Ullah Khan, Samiha Mirza, Alanoud AlOwayed, Fatima M Anis, Reef M Aljuaid, and Reham Baageel. Interpretable machine learning models for malicious domains detection using explainable artificial intelligence (XAI). *Sustainability*, 14(12):7375, 2022.

[11] Akula Ratna Babu, Miriyala Markandeyulu, and Bussa VRR Nagarjuna. Pattern clustering with similarity measures. *Int. J. Computer Technology & Applications*, 3(1):365–369, 2012.

[12] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[13] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11, 2009.

[14] Michael Becher, Felix C Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *2011 IEEE Symposium on Security and Privacy*, pages 96–111. IEEE, 2011.

[15] Dimitris Bertsimas, Agni Orfanoudaki, and Holly Wiberg. Interpretable clustering: an optimization approach. *Machine Learning*, 110:89–138, 2021.

[16] Parnika Bhat and Kamlesh Dutta. A survey on various threats and current state of security in Android platform. *ACM Computing Surveys (CSUR)*, 52(1):1–35, 2019.

[17] Taniya Bhatia and Rishabh Kaushal. Malware detection in Android based on dynamic analysis. In *2017 International conference on cyber security and protection of digital services (Cyber security)*, pages 1–6. IEEE, 2017.

[18] Paula Branco. Exploring the impact of resampling methods for malware detection. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 3961–3968. IEEE, 2020.

[19] Zana Buçinca, Phoebe Lin, Krzysztof Z Gajos, and Elena L Glassman. Proxy tasks and subjective measures can be misleading in evaluating explainable ai systems. In *Proceedings of the 25th international conference on intelligent user interfaces*, pages 454–464, 2020.

[20] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

[21] Gerardo Canfora, Andrea De Lorenzo, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio. Effectiveness of opcode ngrams for detection of multi family Android malware. In *2015 10th international conference on availability, reliability and security*, pages 333–340. IEEE, 2015.

[22] Nicola Capuano, Giuseppe Fenza, Vincenzo Loia, and Claudio Stanzione. Explainable artificial intelligence in cybersecurity: A survey. *IEEE Access*, 10:93575–93600, 2022.

[23] B Chandra and Manish Gupta. An efficient statistical feature selection approach for classification of gene expression data. *Journal of biomedical informatics*, 44(4):529–535, 2011.

[24] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[25] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[26] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 1122–1131. PMLR, 2019.

[27] Tieming Chen, Qingyu Mao, Yimin Yang, Mingqi Lv, Jianming Zhu, et al. Tinydroid: a lightweight and efficient model for Android malware detection and classification. *Mobile information systems*, 2018, 2018.

[28] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426. IEEE, 2013.

[29] Shlomi Dolev, Mohammad Ghanayim, Alexander Binun, Sergey Frenkel, and Yeali S Sun. Relationship of Jaccard and edit distance in malware clustering and online identification. In *2017 IEEE 16th International Symposium on Network Computing and Applications (NCA)*, pages 1–5. IEEE, 2017.

[30] Jeffrey Fairbanks, Andres Orbe, Christine Patterson, Janet Layne, Edoardo Serra, and Marion Scheepers. Identifying attck tactics in Android malware control flow graph through graph representation learning and interpretability. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5602–5608, 2021.

[31] Jianqing Fan and Runze Li. Statistical challenges with high dimensionality: Feature selection in knowledge discovery. *arXiv preprint math/0602133*, 2006.

[32] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. Can we trust your explanations? sanity checks for interpreters in Android malware analysis. *IEEE Transactions on Information Forensics and Security*, 16:838–853, 2020.

[33] Zhiyang Fang, Junfeng Wang, Jiaxuan Geng, and Xuan Kan. Feature selection for malware detection based on reinforcement learning. *IEEE Access*, 7:176177–176187, 2019.

[34] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Fairuz Amalina. Comparative study of k-means and mini batch k-means clustering algorithms in Android malware detection using network traffic analysis. In *2014 international symposium on biometrics and security technologies (ISBAST)*, pages 193–197. IEEE, 2014.

[35] Francisco Fernández-Navarro, César Hervás-Martínez, and Pedro Antonio Gutiérrez. A dynamic over-sampling procedure based on sensitivity for multi-class problems. *Pattern Recognition*, 44(8):1821–1833, 2011.

[36] Ahmad Firdaus, Nor Badrul Anuar, Ahmad Karim, and Mohd Faizal Ab Razak. Discovering optimal features using static analysis and a genetic search based method for Android malware detection. *Frontiers of Information Technology & Electronic Engineering*, 19(6):712–736, 2018.

[37] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[38] Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. Exkmc: Expanding explainable k-means clustering. *arXiv preprint arXiv:2006.02399*, 2020.

[39] Shree Garg, Sateesh K Peddoju, and Anil K Sarje. Network-based detection of Android malicious apps. *International Journal of Information Security*, 16:385–400, 2017.

[40] Daniel Gibert, Carles Mateu, and Jordi Planes. Hydra: A multimodal deep learning framework for malware classification. *Computers & Security*, 95:101873, 2020.

[41] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 364–379, 2018.

[42] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[43] Su Hou, Tianliang Lu, Yanhui Du, and Jing Guo. Static detection of Android malware based on improved random forest algorithm. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 200–200. IEEE, 2017.

[44] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 399–418. Springer, 2016.

[45] Giacomo Iadarola, Fabio Martinelli, Francesco Mercaldo, and Antonella Santone. Towards an interpretable deep learning model for mobile malware detection and family identification. *Computers & Security*, 105:102198, 2021.

[46] Nwokedi Idika and Aditya P Mathur. A survey of malware detection techniques. *Purdue University*, 48(2):32–46, 2007.

[47] Umme Sumaya Jannat, Syed Md Hasnayeen, Mirza Kamrul Bashar Shuhan, and Md Sadek Ferdous. Analysis and detection of malware in Android applications using machine learning. In *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pages 1–7. IEEE, 2019.

[48] Qingshan Jiang, Xinxing Zhao, and Kai Huang. A feature selection method for malware detection. In *2011 IEEE International Conference on Information and Automation*, pages 890–895. IEEE, 2011.

[49] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil DB Bruce, Yang Wang, and Farkhund Iqbal. Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE, 2018.

[50] BooJoong Kang, Suleiman Y Yerima, Kieran McLaughlin, and Sakir Sezer. N-opcode analysis for Android malware classification and categorization. In *2016 International conference on cyber security and protection of digital services (cyber security)*, pages 1–7. IEEE, 2016.

[51] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning*, pages 2668–2677. PMLR, 2018.

[52] Martin Kinkead, Stuart Millar, Niall McLaughlin, and Philip O'Kane. Towards explainable CNNs for Android malware detection. *Procedia Computer Science*, 184:959–965, 2021.

[53] Vasileios Kouliaridis and Georgios Kambourakis. A comprehensive survey on machine learning techniques for Android malware detection. *Information*, 12(5):185, 2021.

[54] Shinho Lee, Wookhyun Jung, Sangwon Kim, and Eui Tak Kim. Android malware similarity clustering using method based opcode sequence and Jaccard index. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 178–183. IEEE, 2019.

[55] Roger J Lewis. An introduction to classification and regression tree (CART) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer, 2000.

[56] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Haipeng Cai, David Lo, and Yves Le Traon. On locating malicious code in piggybacked Android apps. *Journal of Computer Science and Technology*, 32:1108–1124, 2017.

[57] Yuping Li, Jiyong Jang, Xin Hu, and Xinming Ou. Android malware clustering through malicious payload mining. In *Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings*, pages 192–214. Springer, 2017.

[58] Yi-Shan Lin, Wen-Chuan Lee, and Z Berkay Celik. What do you see? Evaluation of explainable artificial intelligence (XAI) interpretability through neural backdoors. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1027–1035, 2021.

[59] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[60] Hui Liu, Qingyu Yin, and William Yang Wang. Towards explainable NLP: A generative explanation framework for text classification. *arXiv preprint arXiv:1811.00196*, 2018.

[61] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun, and Haifeng Liu. A review of Android malware detection approaches based on machine learning. *IEEE Access*, 8:124579–124607, 2020.

[62] Yue Liu, Chakkrit Tantithamthavorn, Li Li, and Yepang Liu. Explainable AI for Android malware detection: Towards understanding why the models perform so well? In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 169–180. IEEE, 2022.

[63] Arvind Mahindru and AL Sangal. Somdroid: Android malware detection by artificial neural network trained using unsupervised learning. *Evolutionary Intelligence*, 15(1):407–437, 2022.

[64] Sapna Malik and Kiran Khatter. System call analysis of Android malware families. *Indian Journal of Science and Technology*, 9(21):1–13, 2016.

[65] Inderjeet Mani and I Zhang. KNN approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.

[66] Luca Massarelli, Leonardo Aniello, Claudio Ciccotelli, Leonardo Querzoni, Daniele Ucci, and Roberto Baldoni. Androdfa: Android malware classification based on resource consumption. *Information*, 11(6):326, 2020.

[67] Sherin Mary Mathews. Explainable artificial intelligence applications in NLP, biomedical, and malware classification: a literature review. In *Intelligent Computing: Proceedings of the 2019 Computing Conference, Volume 2*, pages 1269–1292. Springer, 2019.

[68] Nikola Milosevic, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Machine learning aided Android malware classification. *Computers & Electrical Engineering*, 61:266–274, 2017.

[69] Sina Mohseni, Niloofar Zarei, and Eric D Ragan. A multidisciplinary survey and framework for design and evaluation of explainable AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 11(3-4):1–45, 2021.

[70] Martina Morcos, Hussam Al Hamadi, Ernesto Damiani, Sivaprasad Nandyala, and Brian McGillion. A surrogate-based technique for Android malware detectors' explainability. In *2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 112–117. IEEE, 2022.

[71] Nazifa Mosharrat, Iqbal H Sarker, Md Musfique Anwar, Muhammad Nazrul Islam, Paul Watters, and Mohammad Hammoudeh. Automatic malware categorization based on k-means clustering technique. In *Proceedings of the International Conference on Big Data, IoT, and Machine Learning: BIM 2021*, pages 653–664. Springer, 2022.

[72] Michal Moshkovitz, Sanjoy Dasgupta, Cyrus Rashtchian, and Nave Frost. Explainable k-means and k-medians clustering. In *International conference on machine learning*, pages 7055–7065. PMLR, 2020.

[73] Sunil Kumar Muttoo and Shikha Badhani. Android malware detection: state of the art. *International Journal of Information Technology*, 9:111–117, 2017.

[74] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. Context-aware, adaptive, and scalable Android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):157–175, 2017.

[75] Annamalai Narayanan, Liu Yang, Lihui Chen, and Liu Jinliang. Adaptive and scalable Android malware detection through online learning. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2484–2491. IEEE, 2016.

[76] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7, 2011.

[77] Sang Ni, Quan Qian, and Rui Zhang. Malware identification using visualization images and deep learning. *Computers & Security*, 77:871–885, 2018.

[78] Roland Nilsson, José M Pena, Johan Björkegren, and Jesper Tegnér. Consistent feature selection for pattern recognition in polynomial time. *The Journal of Machine Learning Research*, 8:589–612, 2007.

[79] Ciprian Oprişa, Marius Checicheş, and Adrian Năndrean. Locality-sensitive hashing optimizations for fast malware clustering. In *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 97–104. IEEE, 2014.

[80] Swathi Pai, Fabio Di Troia, Corrado Aaron Visaggio, Thomas H Austin, and Mark Stamp. Clustering for malware classification. *Journal of Computer Virology and Hacking Techniques*, 13:95–107, 2017.

[81] Hamed Haddad Pajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo. Intelligent OS X malware threat detection with code inspection. *Journal of Computer Virology and Hacking Techniques*, 14:213–223, 2018.

[82] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. A systematic literature review of Android malware detection using static analysis. *IEEE Access*, 8:116363–116379, 2020.

[83] Ying Pang, Lizhi Peng, Zhenxiang Chen, Bo Yang, and Hongli Zhang. Imbalanced learning based on adaptive weighting and Gaussian function synthesizing with an application on Android malware detection. *Information Sciences*, 484:95–112, 2019.

[84] David MW Powers. Evaluation: from precision, recall and f-measure to ROC, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020.

[85] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. A survey of Android malware detection with deep neural models. *ACM Computing Surveys (CSUR)*, 53(6):1–36, 2020.

[86] M Zubair Rafique and Juan Caballero. Firma: Malware clustering and network signature generation with mixed network behaviors. In *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings 16*, pages 144–163. Springer, 2013.

[87] Tahsinur Rahman, Nusaiba Ahmed, Shama Monjur, Fasbeer Mohammad Haque, and Muhammad Iqbal Hossain. Interpreting machine and deep learning models for PDF malware detection using XAI and SHAP framework. In *2023 2nd International Conference for Innovation in Technology (INOCON)*, pages 1–9. IEEE, 2023.

[88] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41:77–93, 2004.

[89] Bahman Rashidi, Carol Fung, and Elisa Bertino. Android malicious application detection using support vector machine and active learning. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2017.

[90] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[91] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[92] Maria Sahakyan, Zeyar Aung, and Talal Rahwan. Explainable artificial intelligence for tabular data: A survey. *IEEE Access*, 9:135392–135422, 2021.

[93] Sanjay K Sahay and Ashu Sharma. Grouping the executables to detect malwares with high accuracy. *Procedia Computer Science*, 78:667–674, 2016.

[94] Durmuş Özkan Şahın, Sedat Akleylek, and Erdal Kiliç. Linregdroid: Detection of Android malware using multiple linear regression models-based classifiers. *IEEE Access*, 10:14246–14259, 2022.

[95] Durmuş Özkan Şahin, Oğuz Emre Kural, Sedat Akleylek, and Erdal Kılıç. A novel permission-based Android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, pages 1–16, 2021.

[96] Kanti Sahu and SK Shrivastava. Kernel k-means clustering for phishing website and malware categorization. *International Journal of Computer Applications*, 111(9), 2015.

[97] Sancho Salcedo-Sanz, Laura Cornejo-Bueno, Luís Prieto, Daniel Paredes, and Ricardo García-Herrera. Feature selection in machine learning prediction systems for renewable energy applications. *Renewable and Sustainable Energy Reviews*, 90:728–741, 2018.

[98] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.

[99] Aiman A Abu Samra, Kangbin Yim, and Osama A Ghanem. Analysis of clustering technique in Android malware detection. In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 729–733. IEEE, 2013.

[100] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. Madam: Effective and efficient behavior-based Android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 15(1):83–97, 2016.

[101] Michele Scalas, Davide Maiorca, Francesco Mercaldo, Corrado Aaron Visaggio, Fabio Martinelli, and Giorgio Giacinto. On the effectiveness of system api-related information for Android ransomware detection. *Computers & Security*, 86:168–182, 2019.

[102] The Independent IT security Institute. Avtest.[online]. `https://portal.av-atlas.org/malware`, 2023. 2023.

[103] Fengjun Shang, Yalin Li, Xiaolin Deng, and Dexiang He. Android malware detection method based on Naive Bayes and permission correlation algorithm. *Cluster Computing*, 21(1):955–966, 2018.

[104] Minami Someya, Yuhei Otsubo, and Akira Otsuka. Fcgat: Interpretable malware classification method using function call graph and attention mechanism.

[105] Gautam Srivastava, Rutvij H Jhaveri, Sweta Bhattacharya, Sharnil Pandya, Praveen Kumar Reddy Maddikunta, Gokul Yenduri, Jon G Hall, Mamoun Alazab, Thippa Reddy Gadekallu, et al. XAI for cybersecurity: state of the art, challenges, open issues and future directions. *arXiv preprint arXiv:2206.03585*, 2022.

[106] G Ganesh Sundarkumar and Vadlamani Ravi. Malware detection by text and data mining. In *2013 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–6. IEEE, 2013.

[107] Roopak Surendran, Tony Thomas, and Sabu Emmanuel. A TAN based hybrid model for Android malware detection. *Journal of Information Security and Applications*, 54:102483, 2020.

[108] Michael C Thrun. Distance-based clustering challenges for unbiased benchmarking studies. *Scientific reports*, 11(1):18988, 2021.

[109] Kai Ming Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010.

[110] Ash Turner. How many Android users are there? global and us statistics[online]. `https://www.bankmycell.com/blog/how-many-android-users-are-there`, 2023. 2023.

[111] Anıl Utku, İbrahim Alper Doğru, and M Ali Akcayol. Decision tree based Android malware detection system. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE, 2018.

[112] Victor Van Der Veen, Herbert Bos, and Christian Rossow. Dynamic analysis of Android malware. *Internet & Web Technology Master thesis, VU University Amsterdam*, 106, 2013.

[113] Jasper van der Waa, Elisabeth Nieuwburg, Anita Cremers, and Mark Neerincx. Evaluating XAI: A comparison of rule-based and example-based explanations. *Artificial Intelligence*, 291:103404, 2021.

[114] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks*, 171:107138, 2020.

[115] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating explanation methods for deep learning in security. In *2020 IEEE european symposium on security and privacy (EuroS&P)*, pages 158–174. IEEE, 2020.

[116] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. Deep ground truth analysis of current Android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*, pages 252–276. Springer, 2017.

[117] Zhihua Wen and Vassilios Tzerpos. An effectiveness measure for software clustering algorithms. In *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.*, pages 194–203. IEEE, 2004.

[118] Georg Wicherski. pehash: A novel approach to fast malware clustering. *LEET*, 9:8, 2009.

[119] Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R Lyu. Why an Android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–29, 2021.

[120] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia joint conference on information security*, pages 62–69. IEEE, 2012.

[121] Guoqing Xiao, Jingning Li, Yuedan Chen, and Kenli Li. Malfcs: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *Journal of Parallel and Distributed Computing*, 141:49–58, 2020.

[122] H MutantX-S Xin. Scalable malware clustering based on static features. In *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA, USA*, pages 26–28, 2013.

[123] Sravani Yajamanam, Vikash Raja Samuel Selvin, Fabio Di Troia, and Mark Stamp. Deep learning versus gist descriptors for image-based malware classification. In *Icissp*, pages 553–561, 2018.

[124] Fan Yang, Mengnan Du, and Xia Hu. Evaluating explanation without ground truth in interpretable machine learning. *arXiv preprint arXiv:1907.06831*, 2019.

[125] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):1–40, 2017.

[126] Suleiman Y Yerima, Sakir Sezer, Gavin McWilliams, and Igor Muttik. A new Android malware detection approach using bayesian classification. In *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*, pages 121–128. IEEE, 2013.

[127] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. High accuracy Android malware detection using ensemble learning. *IET Information Security*, 9(6):313–320, 2015.

[128] Jian Yu, Miin-Shen Yang, and E Stanley Lee. Sample-weighted clustering methods. *Computers & mathematics with applications*, 62(5):2200–2208, 2011.

[129] Win Zaw Zarni Aung. Permission-based Android malware detection. *International Journal of Scientific & Technology Research*, 2(3):228–234, 2013.

[130] Zhibo Zhang, Hussam Al Hamadi, Ernesto Damiani, Chan Yeob Yeun, and Fatma Taher. Explainable artificial intelligence applications in cyber security: State-of-the-art in research. *IEEE Access*, 2022.

[131] Yanjie Zhao, Li Li, Haoyu Wang, Haipeng Cai, Tegawendé F Bissyandé, Jacques Klein, and John Grundy. On the impact of sample duplication in machine-learning-based Android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3):1–38, 2021.

[132] Zheng Alan Zhao and Huan Liu. *Spectral feature selection for data mining*. Taylor & Francis, 2012.

[133] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

[134] Hui-Juan Zhu, Zhu-Hong You, Ze-Xuan Zhu, Wei-Lei Shi, Xing Chen, and Li Cheng. Droiddet: effective and robust detection of Android malware using static analysis along with rotation forest model. *Neurocomputing*, 272:638–646, 2018.

[135] Pengfei Zhu, Wangmeng Zuo, Lei Zhang, Qinghua Hu, and Simon CK Shiu. Unsupervised feature selection by regularized self-representation. *Pattern Recognition*, 48(2):438–446, 2015.

[136] Aqil Zulkifli, Isredza Rahmi A Hamid, Wahidah Md Shah, and Zubaile Abdullah. Android malware detection based on network traffic using decision tree algorithm. In *Recent Advances on Soft Computing and Data Mining: Proceedings of the Third International Conference on Soft Computing and Data Mining (SCDM 2018), Johor, Malaysia, February 06-07, 2018*, pages 485–494. Springer, 2018.

Appendix

Appendix A

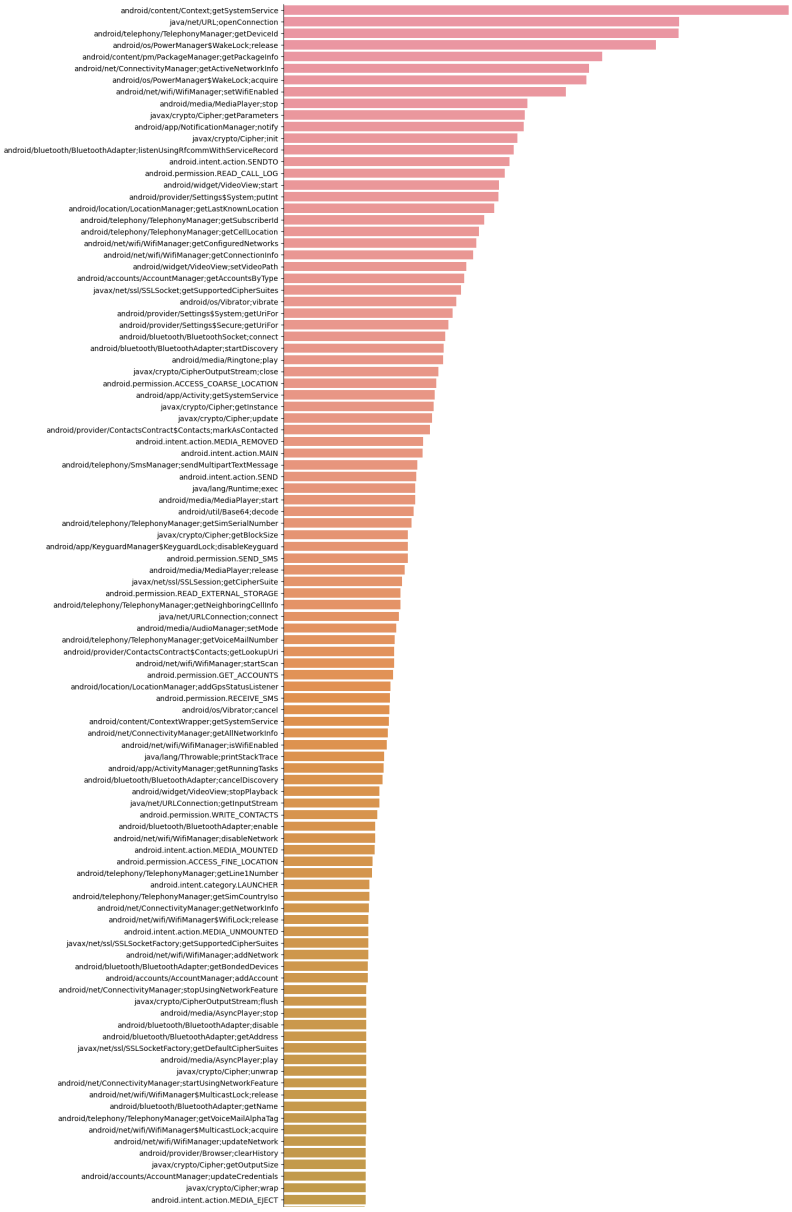


Figure 1: Partial result of feature importance