On the use of machine learning techniques to detect malware in mobile applications

Catarina Palma[1], Artur Ferreira[1,3], and Mário Figueiredo[2,3]

[1] *ISEL, Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa*
[2] *IST, Instituto Superior Técnico, Universidade de Lisboa*
[3] *Instituto de Telecomunicações, Lisboa, PORTUGAL*
`A45241@alunos.isel.pt` `artur.ferreira@isel.pt`
`mario.figueiredo@tecnico.ulisboa.pt`

Abstract. The presence of malicious software (malware), for example in Android applications (apps), has harmful or irreparable consequences to the user and/or the device. Despite the protections provided by app stores to restrict apps containing malware, it keeps growing in both sophistication and diffusion. In this paper, we explore the use of *machine learning* (ML) and *feature selection* (FS) approaches to detect malware in Android applications using public domain datasets. We focus on different data pre-processing, dimensionality reduction, and classification techniques, assessing the generalisation ability of the learned models. The *support vector machine* (SVM) and *random forest* (RF) classifiers achieve the best results, with high accuracy and a low *false negative* (FN) rate. The performance of ML methods is highly dependent on the dataset and its pre-processing and on FS methods identifying the most decisive features to classify an app as malware.

Keywords: Android Applications · Datasets · Feature Selection · Machine Learning · Malware Detection · Security · Supervised Learning.

1 Introduction

The use of smartphones has grown exponentially on the past decade. This growth has been accompanied by the popularisation of Android, an open-source *operating system* (OS) based on the Linux kernel, mainly designed for touchscreen mobile devices. It was first launched in 2008, becoming one of the most popular mobile OS, with 70% of mobile phones using Android, followed by the iPhone OS, with a market share of 28% [15]. Nowadays, we have a huge number of Android applications for different purposes: in the third quarter of 2022, the Google Play Store had approximately 3.5 million apps available [9]. For the most popular apps, the number of users reaches thousands or millions, dealing with a large amount of user-sensitive data. Attackers might target the user's data contained in the smartphone itself via the apps, or they might want to use the device to execute other attacks. Furthermore, from the attacker's perspective, the users are all potential targets and/or victims to download their malware. In a matter of minutes, millions of users can download one app. Malicious attacks against Android mobile devices have increased. In 2020, Kaspersky detected 5.7 million malware Android packages, three times more than in 2019 (2.1 million) [3]. Figure 1 summarises the increase in malware installation packages for smartphone devices from 2017 to 2021. There are software and applications focused on security, such as antiviruses, and major app stores, such as the Google Play Store, also have security and detection mechanisms, such as Google Play Protect, to mitigate malicious apps. These security measures are, to some extent, successful, but malware keeps growing in both sophistication and diffusion, sometimes easily bypassing them. Thus, the need to detect malicious applications is a major issue and *machine learning* (ML) approaches have been proposed to detect malware on Android applications.

In this paper, we explore FS techniques combined with ML for malware detection in Android applications. We emphasize the importance and impact of pre-processing in improving malware detection in Android applications, aspects that have been lacking in many previous studies.

The remainder of this paper is organised as follows. Section 2 provides an overview of the related work for Android malware detection. The proposed approach is described in Section 3. The experimental evaluation is reported in Section 4. Finally, Section 5 ends the paper with concluding remarks and future work directions.

2 Related Work

This section provides an overview of the main topics related to Android malware detection. First, we address the Android OS main blocks (Section 2.1). Then, we discuss the types of malware (Section 2.2). A description of the existing datasets and approaches is provided in Section 2.3. An analysis of existing malware detection techniques is presented in Section 2.4.

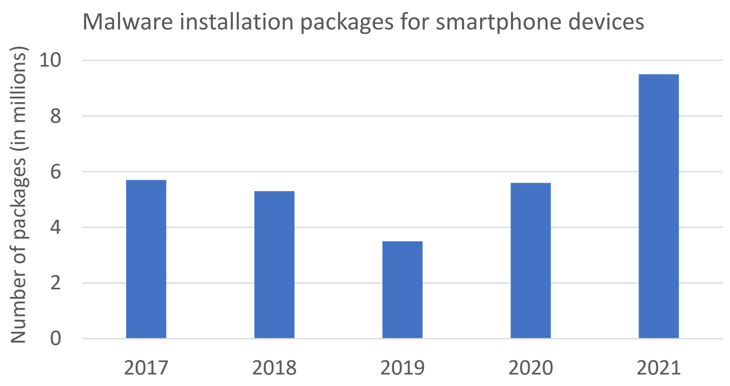

Fig. 1. Malware installation packages for smartphone devices [3].

2.1 Android Operating System

Android is an open-source OS based on the Linux kernel, mainly designed for touchscreen mobile devices. First launched in 2008, it has many versions, with releases taking place every few months. To understand how malware exploits the security vulnerabilities of the Android OS, it is essential to know the basic components of an Android application. Figure 2 shows the components that integrate the *Android package kit* (APK) file of an Android application [17].

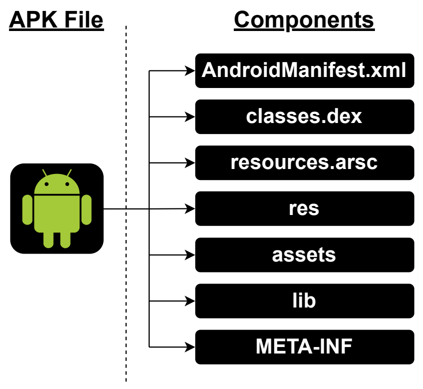

Fig. 2. Android app's components (inspired by Figure 1 in [17]).

Knowledge about the app's components allows understanding the critical security aspects of the Android system. For instance, the applications require system permissions, e.g., access to the camera or the photo gallery to perform specific functionalities. Malware often exploits these accesses and permissions to perform attacks [15]. Thus, the 'AndroidManifest.xml' file, which contains the permissions requested by the application, is of major relevance in determining whether an application is malicious or not.

2.2 Types of Malware

Malware takes different forms in various approaches, such as remote access trojans, banking trojans, ransomware, adware, spyware, scareware, premium text, and others. In general, malware aims to steal sensitive and personal data stored in mobile devices, by exploiting system vulnerabilities such as design weaknesses and security flaws and/or luring the user (e.g., through social engineering) to install applications containing malware [5]. There are well-documented cases of Android malware families, such as "ExpensiveWall", "HummingBad", "FalseGuide", "Judy", and "Chamois", among others, which can be embedded or hidden in many apps on the app stores and then downloaded by millions of users. There are several security measures to mitigate malware attacks, such as using secure internet connections, installing anti-malware apps, and pre-publication validation of the applications by the app stores. Additionally, from the user's perspective, one should keep the apps updated.

Some security measures are inherent to the Android system since the kernel provides key security features, namely, application sandboxing and process isolation [15]. These security measures are widely used and, to some extent, successfully mitigate malware attacks. However, sometimes malware can bypass them: there are a variety of techniques to hinder the identification and neutralisation of malware, e.g., some sophisticated malicious apps can recognise when being executed in emulated environments [14].

2.3 Datasets and Approaches

There are datasets for malware detection in Android applications, such as the Drebin, MalGenome, and VirusShare [17]. These datasets include data ranging from application permissions to *application programming interface* (API) calls. The Drebin dataset was first published in 2014 and contains malware apps from 179 families. Its features are numeric and mostly binary (0 or 1). The CICAndMal2017 dataset was developed by the Canadian Institute and published in 2018 [6]; it contains malware samples from 42 unique malware families, organised into four categories: Adware, Ransomware, Scareware, and SMS Malware. Unlike the Drebin dataset, CICAndMal2017 has both numerical and categorical features.

Kouliaridis and Kambourakis [14] pointed out that regarding feature extraction, there is a stronger focus on source code and the 'AndroidManifest.xml' file analysis and that most publicly available and standard datasets are not recent or up-to-date. Similarly, Muzaffar *et al.* [15] concluded that many existing studies use outdated datasets. Wu *et al.* [17] explored ML-based Android malware detection research papers from January 2019 to November 2020; the authors concluded that the most used features are based on API calls, native opcodes, raw APK files, intents, and permissions. In addition, they provide some insight into the most popular datasets used in the literature, such as Drebin and MalGenome.

Islam *et al.* [12] utilised the CCCS-CIC-AndMal2020 dataset, with 12 major malware categories, 53439 instances, and 141 features. Concerning pre-processing, the authors performed missing data imputation using the "mean" strategy. To deal with class imbalance, the *synthetic minority oversampling technique* (SMOTE) was applied. Additionally, they used the Min-Max method for normalisation and transformed the categorical data into numerical data via one-hot encoding. To perform *feature selection* (FS), the authors applied *recursive feature elimination* (RFE), discarding 60.2% features, achieving 95% accuracy. The authors concluded that the reduced set of features lessened the complexity and improved the accuracy. AlOmari *et al.* [4] proposed a multi-class approach with five classes: Adware, Banking Malware, SMS Malware, Mobile Riskware, and Benign, using the CICMalDroid2020 dataset. The dataset contains 11598 instances and 470 features. Regarding pre-processing, the authors applied z-score normalisation. They used SMOTE and *principal component analysis* (PCA), concluding that SMOTE and z-score normalisation improved the results, while PCA was not beneficial. Keyvanpour *et al.* [13] conducted experiments with the Drebin dataset. In the pre-processing step, the authors applied three FS strategies (effective, high weight, and effective group FS) and concluded that effective FS led to the best results. Alkahtani and Aldhyani [3] applied deep learning algorithms to identify malware on the Drebin and CICAndMal2017 datasets.

2.4 Malware Detection Techniques

To detect malware in Android applications, one of three types of approaches can be followed: static, dynamic, and hybrid. In static analysis, the application is analysed without executing it (in a non-runtime environment). Feature extraction is usually done by code analysis or analysing the 'AndroidManifest.xml' file [14]. Dynamic analysis occurs during the app's regular operation, i.e., when it is running, and is usually performed in monitored, controlled, or sandbox environments [17] to analyse its behaviour. Finally, hybrid analysis combines the previous two types of approach [15].

Wu *et al.* [17] surveyed ML-based Android malware detection papers, from 2016 to November 2020. They concluded that static analysis is the most popular, followed by dynamic and hybrid analysis. Kouliaridis and Kambourakis [14] reached the same conclusion. In fact, static analysis is more popular because it is the simplest approach.

Regarding the use of ML classifiers for Android malware detection, Wu *et al.* [17] found that the most popular classifier was SVM, followed by RF and *k-nearest neighbours* (KNN). Kouliaridis and Kambourakis [14] also analysed several studies, concluding that RFs are the most used among the literature, followed by SVM. Thus, in this paper we consider these classifiers as well as KNN and *naive Bayes* (NB).

AlOmari *et al.* [4] propose a multi-classification approach based on *light gradient boosting model* (LightGBM) using the CICMalDroid2020 dataset. They also analysed the performance of several algorithms, such as KNN, RF, *decision tree* (DT), *linear discriminant analysis* (LDA), and NB. The LightGBM showed the best accuracy and F1-score, achieving 95.49% and 95.47%, respectively.

Islam *et al.* [12] performed multi-classification based on dynamic analysis, with an ensemble ML approach with weighted voting that incorporates RF, KNN, *multi-layer perceptron* (MLP), DT, SVM, and *logistic regression* (LR), with the CCCS-CIC-AndMal2020 dataset. Their proposed weighted voting method showed an accuracy of 95%.

Keyvanpour *et al.* [13] proposed embedding FS in a learning model. The authors opted for effective FS with RF. They also conducted tests with other classifiers, such as KNN and NB. The authors concluded that RF outperformed other models based on several metrics. Based on the FS method, the RF algorithm employs the selected features to detect malicious applications. FS has shown to improve the performance of the RF classifier.

Alkahtani and Aldhyani [3] applied SVM, KNN, and LDA classifiers to identify malware in mobile environments using the Drebin and CICAndMal2017 datasets. SVM achieved the best results with 80.71% accuracy on the Drebin dataset. The results with the CICAndMal2017 dataset were especially positive, with an accuracy of 100%. Overall, it was shown that SVM successfully detects malware. The authors also listed results of other studies that applied RF to the Drebin dataset, obtaining in terms of accuracy, different outcomes such as 98.7%, 94%, 96.7%, and 96%. Some results with SVM also tend to stay within these values, with accuracies of 93.7% (in a 2019 study) and 95% (in a 2021 study) on the Drebin dataset.

Muzaffar *et al.* [15] found that many studies cite high accuracy rates for malware detection. However, several issues with existing approaches may limit their real-world performance. Namely, the use of outdated datasets and inappropriate metrics may give a misleading view of performance.

Shaojie Yang *et al.* [18] proposed an Android malware detection and classification approach based on contrastive learning. Pektaş *et al.* [16] proposed Android malware classification by applying online ML. Adebayo and Aziz [1] proposed an improved malware detection model using the A-priori association rule algorithm. Finally, some authors resort to deep learning techniques [2, 10].

3 Proposed Approach

The approach proposed in this study is depicted in Figure 3. We begin by using a dataset from which we obtain the Android app data, namely the Drebin [7] and CICAndMal2017 [8] datasets. Next, some pre-processing is performed as well as data splitting methods and techniques are applied to properly prepare and organise the data, significantly impacting the model's performance.

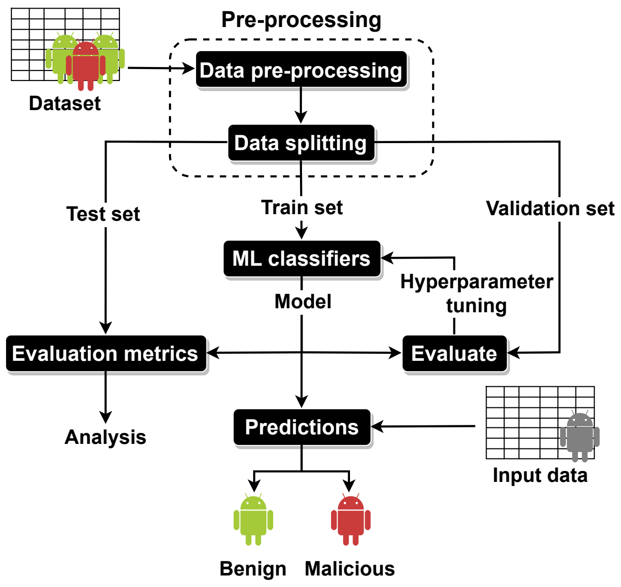

Fig. 3. Block diagram of the proposed approach for Android malware detection.

Three subsets result from the data splitting phase: the train, validation, and test sets. Regarding ML classifiers, this study relies mainly on SVM and RF. Additionally, some experiments with KNN and NB are also performed. With the trained model, input data can be classified as ‘benign’ or ‘malicious’. Thus, it is a binary classification problem.

The next phase is to improve and analyse the model performance. The other two subsets, validation and test, are required for this task. The first one allows evaluating the model to tune the hyperparameters of the ML algorithms to improve the resulting model. The latter enables the analysis of the model through standard evaluation metrics, such as accuracy and confusion matrix. Based on the values reported by these metrics, the methods and techniques used in the data pre-processing and data splitting phases can be improved, thus, enhancing the performance of the trained model in identifying malware in Android applications.

4 Experimental Evaluation

This section reports the experimental evaluation process. The experiments in this study were conducted using the Python programming language and the ML library ‘scikit-learn’. Section 4.1 reports dataset analysis while Section 4.2 describes the evaluation metrics we use to assess the performance of the ML techniques. The experimental evaluation of four well-known classifiers is reported in Section 4.3. Section 4.4 reports experimental results after handling missing values with different approaches. Section 4.5 presents experimental results obtained after applying FS. Finally, Section 4.6 compares some of the experimental results obtained with those from existing studies.

4.1 Dataset Characteristics

The Drebin dataset has n=15036 instances and d=215 features, with no missing values, and all its features are numerical, primarily binary. It has a ratio between class labels of approximately one-third, with the majority belonging to the ‘benign’ class label. The CICAndMal2017 dataset has n=29999 instances and d=183 features. It contains missing values (around 200) and numerical and categorical features. It also has a ratio between class labels of approximately one-third, but, as opposed to the Drebin dataset, the majority belongs to the ‘malicious’ class label. Thus, both datasets are not balanced, and the data from CICAndMal2017 requires more pre-processing than Drebin’s data. The CICAndMal2017 dataset contains categorical features and missing values, both incompatible with the SVM algorithm.

4.2 Evaluation Metrics

In this section, we describe the evaluation metrics used to assess the performance of the ML models. We consider two target classes: ‘benign’ and ‘malicious’. A *true positive* (TP) means to classify a malicious app as malicious correctly, a *true negative* (TN) is to classify a benign app as benign, a *false positive* (FP) is to classify a benign app as malicious and a *false negative* (FN) refers to classifying a malicious app as benign. When assessing performance, we aim to minimise type 1 (FP) and type 2 (FN) errors. However, we consider the type 2 errors to be more problematic since it is preferable to label a benign app as malicious rather than to indicate that the app is safe when it has malicious code leading to compromised security. The accuracy (Acc) measures the proportion of correct classifications out of all predictions and is given by

$$Acc = \frac{TN + TP}{TN + TP + FN + FP}. \tag{1}$$

Although we aim for high accuracy, this can be misleading when working with imbalanced data, with some disparity in the proportion of instances per class. We have checked that the Drebin and CICAndMal2017 datasets have some imbalances in the data. For this reason, in our experiments, we have also assessed the Recall (Rec) metric,

$$Rec = \frac{TP}{TP + FN}, \tag{2}$$

also known as the true positive rate or sensitivity.

4.3 Experimental Results: Baseline

First, basic data pre-processing was required to apply the different ML classifiers, RF, SVM, KNN and NB, to the Drebin and CICAndMal2017 datasets. We converted each categorical feature to numeric through label encoding. Additionally, to initially deal with missing values, instances containing them were removed.

Regarding the data splitting phase, a first, simple and traditional machine learning approach was taken by random splitting the data with a ratio of 70-30 for training and testing, respectively. The first experimental results were obtained with the evaluation metrics, accuracy, confusion matrix, and recall for each ML classifier with each dataset, as shown in Table 1.

The four classifiers showed promising results in detecting malware on the Drebin dataset, with RF providing the best result in terms of accuracy (98.60%). However, experiments using the CICAndMal2017 dataset showed mostly unsatisfactory results, with only the RF classifier achieving a reasonable accuracy of 80.49%. For the recall metric, the classifiers also presented good results on the Drebin dataset, since the number of FN was below 100 for each classifier. On the CICAndMal2017 dataset, we obtained the best recall value of 99.76% with the SVM classifier and the worst of 80.84% with the KNN classifier. Although the SVM classifier got the best result with the CICAndMal2017 for the recall metric, in terms of accuracy it only obtained 65.82%. Thus, as seen in the confusion matrix, it presents a low value of FN but a very high FP rate.

On the Drebin dataset, the FN occurrences tend to occur more than FP, likely because the Drebin dataset has more instances of benign apps than malicious

Table 1. Experimental results with the evaluation metrics Acc, confusion matrix elements (TN, FP, FN, and TP), and Rec for each ML classifier on each dataset.

Classifier	Dataset	Acc (%)	TN	FP	FN	TP	Rec (%)
RF	Drebin	98.60	2814	13	50	1634	97.03
RF	CICAndMal2017	80.49	2060	930	781	5001	86.49
SVM	Drebin	97.94	2805	22	71	1613	95.78
SVM	CICAndMal2017	65.82	6	2984	14	5768	99.76
KNN	Drebin	97.58	2782	45	64	1620	96.20
KNN	CICAndMal2017	64.00	940	2050	1108	4674	80.84
NB	Drebin	93.08	2611	216	96	1588	94.30
NB	CICAndMal2017	65.50	461	2529	497	5285	91.40

ones. For the CICAndMal2017 experiments, there are more FP occurrences than FN since it has a higher number of malicious apps. Additionally, since both the RF and SVM are the classifiers presenting overall better results for this problem, as was also stated in the surveyed works, our following experiments proceeded with the use of both these methods.

Using the 'GridSearchCV' function, we aimed to tune the hyperparameters of the classifiers to improve the results on both datasets. Namely, we tuned the most relevant parameters in each classifier, such as the number of trees in RF and the C parameter in SVM. The tuning of the ML classifier's hyperparameters slightly improved the efficiency of the trained model, but this improvement didn't rise more than 1% in terms of accuracy. Hyperparameter tuning might provide a better result for one dataset but worse for another.

4.4 Experimental Results: Handling Missing Values

After hyperparameter tuning, we aimed to improve the data pre-processing stage, namely, how we handle missing values in the CICAndMal2017 dataset. The first approach was to remove the instances containing them. Table 2 reports these experimental results.

Table 2. Experimental results with the evaluation metrics Acc, confusion matrix elements (TN, FP, FN, and TP), and Rec, for the RF and SVM classifiers on the CICAndMal2017 dataset, for each of the methods chosen to impute missing values.

Classifier	Method	Acc (%)	TN	FP	FN	TP	Rec (%)
RF	Removing instances with missing values	80.55	2074	916	790	4992	86.33
RF	Removing features with missing values	80.88	2089	883	838	5190	86.1
RF	Replacing missing values with the mean	81.06	2088	884	821	5207	86.38
SVM	Removing instances with missing values	65.74	219	2771	234	5548	95.95
SVM	Removing features with missing values	67.28	419	2553	392	5636	93.5
SVM	Replacing missing values with the mean	67.07	373	2599	365	5663	93.94

In general, the obtained results for each method do not vary substantially. With the RF classifier, the method of dealing with missing values by replacing them with the median achieved the best result. While for the SVM classifier, removing features and instances containing missing values provided better results in terms of accuracy and recall, respectively. The latter result, plus the fact that the results after removing whole instances or features (containing missing values) do not differ significantly from the ones where we replace missing values with the estimated mean value, is an indicator that the CICAndMal2017 dataset possesses data that is irrelevant, maybe even harmful, for the training of the model. Thus it would be adequate to perform dimensionality reduction by using FS techniques, for example.

4.5 Experimental Results: Feature Selection

Following the previous conclusions, we aimed to apply FS to the datasets. However, before using any FS technique, we deemed some more pre-processing necessary. Namely, in the CICAndMal2017 dataset, five features were categorical and, thus, converted to numerical. However, since we are using label encoding, there are discrepancies between the values. Min-Max normalisation was applied to accommodate all the values of both datasets between 0 and 1.

After normalisation, the results improved in the CICAndMal2017 dataset, with the accuracy achieved by SVM rising from 67.07% to 71.42%. No impact was seen in the case of the Drebin dataset since its values were binary (thus, unaffected by the normalisation applied).

Finally, for FS, we opted to apply the *relevance-redundancy FS* (RRFS) filter [11]. As relevance measure, we used the Fisher ratio, and as a redundancy measure, we used the *absolute cosine* (AC). Table 3 summarises, in terms of accuracy, the baseline and experimental results after applying RRFS.

Aside from the SVM classifier on the CICAndMal2017 dataset, which improved substantially from 65.82% to 70.42%, the remaining results do not seem to vary much and sometimes even worsen slightly, as is the case with the Drebin dataset. However, these slight drops in accuracy in some of the results are arguably compensated by the reduction of the number of features. Table 4 presents the original numbers of features versus the numbers of features after applying the RRFS approach.

In both datasets, the number of features was significantly reduced, and, as seen in Table 3, the accuracy obtained only differed slightly, remaining a good result while being able to reduce the number of features substantially. Besides this, the RRFS approach also allows insight into the most relevant

Table 3. Experimental results at baseline and after FS with the Acc (%) evaluation metric for the RF and SVM classifiers on the Drebin and CICAndMal2017 datasets.

Classifier	Dataset	Baseline	RRFS
RF	Drebin	98.60	96.92
RF	CICAndMal2017	80.49	81.42
SVM	Drebin	97.94	96.36
SVM	CICAndMal2017	65.82	70.42

Table 4. Number of features (d) for the Drebin and CICAndMal2017 datasets, before and after applying the RRFS approach [11].

Dataset	d (Original)	d (after RRFS)
Drebin	215	94
CICAndMal2017	183	64

features on each dataset to classify malware. The five most relevant features according to the RRFS approach to classify malware on the Debrin dataset are the following: `transact`, `SEND_SMS`, `Ljava.lang.Class.getCanonicalName`, `android.telephony.SmsManager` and `Ljava.lang.Class.getField`.

Keyvanpour *et al.* [13] also applied FS with effective and high weight FS, and reported the most relevant features on the Drebin dataset. Two of them coincide with our five most relevant features: `SEND_SMS` and `android.telephony.SmsManager`.

4.6 Experimental Results: Comparative Analysis of Results

Since the datasets herein used, Drebin and CICAndMal2017, are also used by Alkahtani and Aldhyani [3], we will briefly compare our results with theirs. Similarly to our approach, those authors randomly divided the datasets into 70% for training and 30% for testing. Regarding pre-processing, they only mention Min-Max normalisation. Aside from this, no other pre-processing methods or tuning of hyperparameters are mentioned. Thus, the methodology with which the results were obtained differs from ours. Since the authors did not use the RF classifier, we will compare only the accuracy results regarding SVM. Table 5 summarises these results.

The proposed approach presented better accuracy on the Drebin dataset, achieving 96.36% accuracy compared to 80.71% reported by Alkahtani and Aldhyani [3]. However, regarding the CICAndMal2017 dataset, the proposed approach only achieved 70.42% compared to the accuracy of 100% that those authors claim to have achieved. This disparity in the obtained results between the two studies using the same datasets lies in the different methodologies applied, namely, the pre-processing applied, which significantly impacts the obtained results.

Table 5. Comparison of the experimental results, in terms of accuracy (%), obtained by Alkahtani and Aldhyani [3] with the SVM classifier with the ones obtained with the proposed approach using the same classifier.

Dataset	Alkahtani and Aldhyani	Proposed
Drebin	80.71	**96.36**
CICAndMal2017	**100.00**	70.42

5 Conclusions

Malware in Android applications affects millions of users worldwide and is constantly evolving. Thus, its detection is a current and relevant problem. In the past few years, ML approaches have been proposed to mitigate malware in mobile applications by performing supervised classification. In this study, an approach was proposed, and experiments were performed to explore the ML methods and techniques that effectively mitigate this problem. Namely, we performed experiments with the RF, SVM, KNN, and NB classifiers and then, with some more experimental results, we aimed to improve the trained model.

Based on the surveyed works, we concluded that the RF and SVM classifiers presented better results, thus, being the most popular. There are a variety of publicly available datasets, with one of the most popular being the Drebin dataset, but many seem not to be up-to-date. Static analysis is the most used throughout the existing approaches regarding the analysis type. As for evaluation metrics, the accuracy metric is commonly the most used. Lastly, disclosing the data pre-processing methods and techniques used seems to be absent in many existing studies. Often authors reference the data splitting method (usually random split) but do not specify any method or technique for data pre-processing, making it difficult to reproduce the experiments and compare results accurately.

Our experimental results show that the RF and SVM classifiers perform best in this problem. The experiments using the CICAndMal2017 dataset showed unsatisfying results, most likely because it requires more pre-processing than the Drebin dataset. Namely, it contains missing values and categorical features, while the Drebin dataset presented no missing values and all its features were already numerical. Furthermore, we also concluded that data pre-processing and hyperparameter tuning of the ML classifiers greatly impacts the performance of the trained model. However, they might provide a better result for one dataset but a worse outcome for another. Thus, there is no ideal solution for all datasets. Hence, experiments with more datasets should take place.

In future work, more attention should be given to the data pre-processing and dimensionality reduction stages. Some other evaluation metrics can be considered, such as F1-score and area under the curve, to further evaluate ML approaches in the detection of malware in Android applications.

References

[1] Olawale Surajudeen Adebayo and Normaziah Abdul Aziz. "Improved malware detection model with apriori association rule and particle swarm optimization". In: *Security and Communication Networks* 2019 (2019). DOI: `https://doi.org/10.1155/2019/2850932`.

[2] Muhammad Shoaib Akhtar and Tao Feng. "Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time". In: *Symmetry* 14.11 (2022). ISSN: 2073-8994. DOI: `10.3390/sym14112308`. URL: `https://www.mdpi.com/2073-8994/14/11/2308`.

[3] Hasan Alkahtani and Theyazn HH Aldhyani. "Artificial intelligence algorithms for malware detection in Android-operated mobile devices". In: *Sensors* 22.6 (2022), p. 2268.

[4] Hani AlOmari, Qussai M. Yaseen, and Mohammed Azmi Al-Betar. "A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection". In: *Procedia Computer Science* 220 (2023). The 14th International Conference on Ambient Systems, Networks and Technologies Networks (ANT 2022) and The 6th International Conference on Emerging Data and Industry 4.0 (EDI40), pp. 763–768. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2023.03.101. URL: https://www.sciencedirect.com/science/article/pii/S1877050923006361.

[5] 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)*. IEEE. 2019, pp. 110–117.

[6] *Android Malware 2017 — Datasets — Research — Canadian Institute for Cybersecurity — UNB*. https://www.unb.ca/cic/datasets/andmal2017.html. (Accessed on 27/02/2023).

[7] *Android Malware Dataset for Machine Learning — Kaggle*. https://www.kaggle.com/datasets/shashwatwork/android-malware-dataset-for-machine-learning. (Accessed on 27/02/2023).

[8] *Android Permission Dataset — Kaggle*. https://www.kaggle.com/datasets/saurabhshahane/android-permission-dataset. (Accessed on 27/02/2023).

[9] *Biggest app stores in the world 2022 — Statista*. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. (Accessed on 13/02/2023).

[10] Omar N. Elayan and Ahmad M. Mustafa. "Android Malware Detection Using Deep Learning". In: *Procedia Computer Science* 184 (2021). The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops, pp. 847–852. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2021.03.106. URL: https://www.sciencedirect.com/science/article/pii/S1877050921007481.

[11] Artur Ferreira and Mário Figueiredo. "Efficient feature selection filters for high-dimensional data". In: *Pattern Recognition Letters* 33.13 (2012), pp. 1794–1804. ISSN: 0167-8655. DOI: http://dx.doi.org/10.1016/j.patrec.2012.05.019. URL: http://dx.doi.org/10.1016/j.patrec.2012.05.019.

[12] Rejwana Islam et al. "Android malware classification using optimum feature selection and ensemble machine learning". In: *Internet of Things and Cyber-Physical Systems* 3 (2023), pp. 100–111. ISSN: 2667-3452. DOI: https://doi.org/10.1016/j.iotcps.2023.03.001. URL: https://www.sciencedirect.com/science/article/pii/S2667345223000202.

[13] Mohammad Reza Keyvanpour, Mehrnoush Barani Shirzad, and Farideh Heydarian. "Android malware detection applying feature selection techniques and machine learning". In: *Multimedia Tools and Applications* 82.6 (2023), pp. 9517–9531. DOI: `https://doi.org/10.1007/s11042-022-13767-2`.

[14] Vasileios Kouliaridis and Georgios Kambourakis. "A comprehensive survey on machine learning techniques for Android malware detection". In: *Information* 12.5 (2021), p. 185.

[15] Ali Muzaffar et al. "An in-depth review of machine learning based Android malware detection". In: *Computers & Security* (2022), p. 102833.

[16] Abdurrahman Pektaş, Mahmut Çavdar, and Tankut Acarman. "Android Malware Classification by Applying Online Machine Learning". In: *Computer and Information Sciences*. Ed. by Tadeusz Czachórski et al. Cham: Springer International Publishing, 2016, pp. 72–80.

[17] Qing Wu, Xueling Zhu, and Bo Liu. "A survey of Android malware static detection technology based on machine learning". In: *Mobile Information Systems* (2021).

[18] Shaojie Yang et al. "An Android Malware Detection and Classification Approach Based on Contrastive Lerning". In: *Computers & Security* 123 (2022), p. 102915. ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2022.102915`. URL: `https://www.sciencedirect.com/science/article/pii/S016740482200308X`.

References

1. Ferreira, A. and Figueiredo, M. (2012). Efficient feature selection filters for high-dimensional data. *Pattern Recognition Letters*, 33(13):1794 – 1804.
2. Turner, A. Android vs. Apple Market Share: Leading Mobile Operating Systems (OS). `https://www.bankmycell.com/blog/android-vs-apple-market-share/`. Last accessed Jun 2023
3. Alkahtani, H. & Aldhyani, T. Artificial intelligence algorithms for malware detection in Android-operated mobile devices. *Sensors*. **22**, 2268 (2022)
4. Elayan, O. & Mustafa, A. Android Malware Detection Using Deep Learning. *Procedia Computer Science*. **184**, 847-852, (2021)
5. Akhtar, M. & Feng, T. Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. *Symmetry*. **14**, 2308, (2022)
6. Wu, Q., Zhu, X. & Liu, B. A survey of Android malware static detection technology based on machine learning. *Mobile Information Systems*. (2021)
7. Muzaffar, A., Hassen, H., Lones, M. & Zantout, H. An in-depth review of machine learning based Android malware detection. *Computers & Security*. pp. 102833 (2022)
8. Alqahtani, E., Zagrouba, R. & Almuhaideb, A. A Survey on Android Malware Detection Techniques Using Machine Learning Algorithms.. *2019 Sixth International Conference On Software Defined Systems (SDS)*. pp. 110-117 (2019)
9. Kouliaridis, V. & Kambourakis, G. A comprehensive survey on machine learning techniques for Android malware detection. *Information*. **12**, 185 (2021)
10. Yang, S., Wang, Y., Xu, H., Xu, F. & Chen M., An Android Malware Detection and Classification Approach Based on Contrastive Learning. *Computers & Security*. **123**, pp. 102915 (2022)

11. Pektaş, A., Çavdar, M. & Acarman, T. Android Malware Classification by Applying Online Machine Learning. *Computer And Information Sciences*. pp. 72-80 (2016)
12. Adebayo, O. & Abdul Aziz, N. Improved Malware Detection Model with Apriori Association Rule and Particle Swarm Optimization. *Security And Communication Networks*. **2019** pp. 2850932 (2019,8), https://doi.org/10.1155/2019/2850932
13. Keyvanpour, M., Barani Shirzad, M. & Heydarian, F. Android malware detection applying feature selection techniques and machine learning. *Multimedia Tools And Applications*. **82**, 9517-9531 (2023)
14. AlOmari, H., Yaseen, Q. & Al-Betar, M. A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection. *Procedia Computer Science*. **220** pp. 763-768 (2023)
15. Islam, R., Sayed, M., Saha, S., Hossain, M. & Masud, M. Android malware classification using optimum feature selection and ensemble machine learning. *Internet Of Things And Cyber-Physical Systems*. **3** pp. 100-111 (2023)
16. Statista, `https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/`. Last accessed Jun 2023
17. UNB, `https://www.unb.ca/cic/datasets/andmal2017.html`. Last accessed Jun 2023
18. Kaggle, Android Malware Dataset for Machine Learning, `https://www.kaggle.com/datasets/shashwatwork/android-malware-dataset-for-machine-learning` Last accessed June 2023
19. Kaggle, Android Permission Dataset, `https://www.kaggle.com/datasets/saurabhshahane/android-permission-dataset` Last accessed Jun 2023