FG-Droid: Grouping Based Feature Size Reduction for Android Malware Detection through Machine Learning

Recep Sinan ARSLAN (sinanarslanemail@gmail.com)
Kayseri Üniversitesi https://orcid.org/0000-0002-3028-0416

Research Article

Keywords: Permission grouping, size reduction, Android OS, security, malware detection

Posted Date: December 22nd, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1126907/v1

Version of Record: A version of this preprint was published at PeerJ Computer Science on July 14th, 2022. See the published version at https://doi.org/10.7717/peerj-cs.1043.

Abstract

The number of applications prepared for use on mobile devices has increased rapidly with the widespread use of the Android OS. This has resulted in the undesired installation of Android apks that violate user privacy or malicious. The increasing similarity between Android malware and benign applications makes it difficult to distinguish them from each other and causes a situation of concern for users. In this study, FG-Droid, a machine-learning based classifier with an efficient working system, using the method of grouping the features obtained by static analysis, was proposed. It was created as a result of experiments with Machine learning (ML), DNN, RNN, LSTM and GRU based models using Drebin, Genome and Arslan datasets. Experimental results reveal that FG-Droid has achieved 97.7% AUC score with a vector includes only 11 static features, and ExtraTree algorithm. FG-Droid analyze the applications with using the least number of features compare to previous studies, and required the least processing time for training and prediction. As a result, it has been shown that Android malware can be detected in high accuracy rate with an effective feature set and there is no need to use a large number of features extracted with different techniques (static, dynamic or hybrid).

1. Introduction

Android OS is a mobile platform and has been prepared by a group of developers. It has the appropriate functional infrastructure to access hardware resources. It is free and open source platform and is equipped with a security framework that relies on the Linux kernel [1]. However, since its security structure is based on the application layer [5], these devices becomes partially or completely exposed to numerous security attacks, making them a routine target [4]. In addition, the widespread use of mobile devices with Android OS, the adoption of these devices by end users and the resulting of increasing market share, has caused it to become the target of cyber hackers, especially web based and application layer based attacks [2–3].

When malicious applications Access user mobil devices, they can engage in a series of malicious activities such as obtaining confidential information, seizing more authorized user accounts, and misusing the obtained certain level of security [4]. Google Play has set tu a permission-based system to control applications' access to confidental data. This permission is asked to users before installation, taking into account the resources of the application. Users must approve these permissions before instalation to use it. However, this-preapproval mechanism does not provide suffiecient protection for users, since users accept these permissions without detailed examination. As a result, users accept all conditions in order to have free access to applications that offer the features they demand [6]. For these reasons, there is a need to work on the security mechanism that Google Play leaves to users.

Malware detection mechanism in different types have been proposed to address this need in the security mechanism. These are signature-based approach [7, 8], behaviour based detection software [9–11] and machine learning based approaches [12–14]. Among these, machine learning and deep learning architectures have gain more popularity recently. Because in this method, it is possible to obtain both a

dynamic learning and development process and good results against zero-day attacks. It is possible to create ML models that are open to learning and development at the same speed for cyber attackers who try to overcome the security mechanism with a new technique every day, and to produce promising solutions for the detection of malware [15]. In machine learning based approaches, there is a dependency on the features used as input, classifier and learning architectures. In this study, the size of the feature set and the effects of the features it contains in terms of performance and efficienty in android malware detection mechanisms are focused.

There are many manual and automatic feature extraction methods. These methods are basically divided into 3 groups as static features, dynamic features and hybrid features [16]. Each of these features can be used to detect different malicious acitivites. In addition, each feature can provide a distinctiveness in detecting malicious applications. For this reason, different feature sets can be used in studies in this field, and problem-specific feature vectors can be produced. The main purpose is to provide fastest and highest classification success model with the best feature set.

The contributions of this work in the following:

- Non-essential feature selection and feature grouping: FG-Droid uses a feature grouping approach, unlike all existing method for feature selection. Thus, an efficient and effective classifier with a low-dimensional feature set has been revealed.
- The proposed FG-Droid model has been tested by conducting extensive experiments on malicious and benign apks wit widely used real data. By using comparative test environments, both the classification performance of the proposed model and training and prediction times required for classification were measures. As a result of the evaluation, is has been shown that the while the size reduction and selection methos in the feature vector based on feature grouping is successfull in the classification results, it is quite efficient in terms of the processin time.

The continuation of the study is organized as follows: In Section 2, Android application development infrastructure and similar studies on this subject are analyzed separately for static and dynamic methos. The methodology of FG-Droid is explained in detail in Section 3 and the experimental results are given comparatively in Section 4. In the last part, a general evaluation of the study is made and suggestions were made for future studies. In addition, FG-Droid permission grouping table is given as Appendix-1 after the reference section.

2. Android Application Structure And Related Works

In this section, after explainin the Androic application architecture, a literature reviwe on Android malware detection tools, in which machine learning techniques were used as classifiers, is given. Different feature extraction methods were used to reach the feature sets. These methods can basically be grouped into 3 groups: static, dynamic and hybrid [29]. Studies on the subject was given by combining them under these groups.

2.1. Android Application Structure

Applications prepared fort he Android OS are presented to users as a kind of compressed file with APK extension. The package includes files such as souce code, manifest.xml, libraries, resources, dex file, properties as shown in Figure-1. Applications are developed in Java using the Android SDK. Applications whose source code is completed are converted to Dalvik bytecode (Dex) together with other required files. A manifest file is an xml that contains basic cookies for the applications, links to external files and libraries such as activicies, receivers, and content providers. In addition, the permissions needed to access device resources, target platform data are included in the manifest. External resources such as videos, sounds, audios, images, text files used by the application are packaged in the apk file. As a result, a package is prepared that contains necessary files to execute the entire working functions [51]. Prepared apk packages are offered to users via Google Play Sotre or third party application distribution platforms. More details for Android application development are provided on the developer page [50].

2.2. Static Analysis

The features extracted as a result of the static analysis are basically obtained from the Androidmanifest.xml in the apk package the SMALI files. It is intended to extract features without actually running applications and use them for classification. Permissions, API calls, intent filters, application metadata, function calls, opcode are some of these features. Among these, permission and API calls are more commanly used.

Permissions are one of the most researched and widely used features in malware analysis, as they form the basis of the android security architecture. Android applications were installed on the mobile devices after approval from the user. Since permission is the first obstacle for cyber hackers to reach their malicious targets, many researchers [17–19] have carried out permission-based analysis studies. Syrris et al. [24] examined malware detection based on machine learning and static analysis. The six best-known ML techniques were evaluated both in terms of classification rate and feature selection. In the experimental results, it has been shown that high accuracy rate can be achieved with a lower dimensional feature set, as in FG-Droid. Anastasia [25] is a classifier using machine learning (dt, rf, svm etc.) and deep neural network. Static analyis tool has been created to extract features such as intent, system commands, permissions and api-calls. In the tests performed with a dataset consisting of 11187 benign and 18677 malicious applications, a TP value of 97.3% was reached, and the f-score was 96.0%. Droidmat[27] is another tools that uses permissions, intents, api calls and services to classify android apks. Analysis was performed using the K-neighbors algorithm. As a result, 91.82% f1 score was obtained. Drebin [29] works with a large dimensional static feature vector, since limited hardware resources pose problems for dynamic analysis. The extracted feature vector was trained with SVM and a classification success of 94% has been achieved. It has been stated that an average of 10 seconds is required for an analysis.

API calls is another feature that can be obtained as a results of static analysis. Because applications need API callas to communicate with the device. Therefore, API calls can be useful to understand the intent of applications. Many researchers [20–23] have targeted malware detection using API calls. High classification success has been achieved in studies with API calls, but there are approximately 32000 different APIs on the Android platform [15]. This causes the feature vector to be very large size. In addition, a very limited part of these feature were used by applications. DroidAPIMiner [26] uses package-level parameters in the feature vector to catch malicious API calls. As a result, 2.2% FP rate was achieved. MAMADroid [28, which analyzes the API usage behavior of applications, is a more robust system for malware detection. It has been stated that the model has shown high success rate in tests for many years. Taheri et al. [42] similarly used the feature vector produced by using API calls, intents and permission from training and prediction phases for 4 different classifiers. Similarity calculation was made using Hamming distance and an accuracy value of 91% was achieved. Static analysis method have been used in many other studies such as DroidSieve [40], DroidDet[41].

Although, static analysis is advantageous at certain points, is also has some limitations [33]. It would not be possible to observe its behavior at runtime. Code analysis of complex software takes time. It may not be possible to extract static features based on the source code inn encrpted and obsfucated applications.

2.3. Dynamic Analysis

Dynamic features are extracted as a result of analyzing the situations in which Android application communicate with the operating systems or the network. System calls and network usage statistics are the most basic features that give an idea about the apks. In addition, processor and memory space usage data, that status of the services running instantaneously, some statistical data about the device (battery usage, screen-on time etc.), and information about the addresses reached by the systems calls and network packets. Droidscope [30] is an efficient and effective dynamic android malware detection tool that works on Android devices by extracting three-layer (hardware, operating system and dalvik virtual machine) system calls, and performing semantic analysis at the operating system and code level. MADAM [31] tracks kernel-level system calls and user-level usage statistics and activities to be able to describe and classify the behavior of a mobile application. ANDLANTIS [35] is a dynamic analysis tool that runs on a sandbox. It aims to detect malware by analyzing system calls, footprints, running behaviors. It requires 1 hour for the analysis of 3000 applications. Chen et al. [39] proposed a semi-supervised classifier that works using dynamic API usage logs. They make use of both labeled and unlabeled data to obtain application properties. They showed the results comparatively by classifying with SVM and KNN.

Dynamic analysis-based approaches try to understand the intentions of mobile applications by analyzing their behavior during running. For this reason, in some cases, it can show higher recognition success than static analysis. However, in order to understand a suspicious behavior, it must be run at least once and information must be collected during this time. This means both creating a security problem for the device and additional processing time [32]. The presence of processor and memory limitations of mobile devices complicates the applicability of dynamic analysis. However, running applications on an

emulator/sandbox may be sufficient for dynamic analysis. It was accepted that it is possible to gather more information about the application by simulating the so-called user behavior [34].

A single feature can capture certain aspects of an application. However, using more than one feature together can be more advantageous in malware detection. Various studies have been conducted with hybrid feature structures using combinations of both static and dynamic features [36–38]. ProfileDroid [43] evaluates both static and dynamic features such as Android permissions, features obtained as a result of code analysis, network usage statistics. Thus, a systematic study was aimed to establish a cost-effective and consistent model.

3. Methodology

In this section, the method used by the FG-Droid tool was explained in detail. The method includes the stages of extracting features, grouping extracted features, updating feature values, and testing on sample dataset.

3.1. Automatic Feature Extraction and Pre-processing

In the feature extraction stage, each application was made ready to be used in the learning model by going through the processing steps in Figure 2. The apk files in the malicious and benign application datasets are made accessible to the package content by using aapt (android asses packaging tool) module in the first stage. After this conversion process, the manifest.xml file containing the permissions information in the apk package was parsed. The aim is to determine the requested permissions. A feature vector array was created for each application and saved as one line in the csv file. While adding to the csv file, malicious and benign applications were labeled. In the final stage, the labeled data was formatted and made available for learning.

The processing steps shown in Figure-2 were carried out by means of an automatic code, and it is a very fast feature extraction process. At this stage, the suggested feature grouping and reduction was not done, and a dataset containing 349 features was obtained for all applications.

After this process, preprocessing steps were applied to achieve the best classification performance. These steps include removal of NaN data, deletion of duplicate samples, and normalization. The normalization process was performed using the normal distribution and the values were scaled to the range of 0-1. The normalization process was performed after the feature grouping process.

3.2. Proposed Feature Grouping and Feature Selection Algorithm

The mechanism of accessing certain components or performing functions was based on permissions in the Android operating system. Android applications request permission to access and read information about calendar, location information, contacts, storage, camera, microphone, various sensors. For

example, it is possible to access contact information on the device with the Android.permission.READ_CONTACT permission. Permissions vary according to the device's feature and functional capacity.

In this study, instead of a feature vector containing all permissions used in Android architecture, a model called FG-Droid has been developed, which can achieve high classification success with a very fast training and testing time by using fewer features. For this purpose, a series of operations were carried out within the flow chart shown in Figure-3.

In the first stage of the model, a dataset containing 7622 applications was created, the details of which are given in section-4.The permission-based features of these applications have been extracted. At this stage, a training and test vector containing 349 features was created. In order to shorten the training and testing times and to classify using less processing time, operations were carried out according to the proposed algorithm for grouping the feautures and reduction the size. In this way, a tool was created in which it is possible to classify with less features. The standardization process was performed using the normal distribution on the feature vector obtained after this step. The resulting data set was divided into two separate parts to be used in the training and testing phases. During the training phase, oversampling was performed using the SVMSMOTE algorithm to resolve the sample number imbalance between clusters, and training was carried out on DNN, RNN, LSTM and GRU networks together with machine learning algorithms (Rf, Extratree, gradient boosting, etc.). The resulting trained models were tested with previously separated test data and the results are shown in comparison with commonly used metrics.

As can be seen in Figure-3, the proposed feature grouping and selection method wasa carried out after the application features are extracted. Thus, it has been possible to work with a much lower-dimensional feature vector not only in the training and testing phases, but also in all processing steps. The details of feature grouping operations are as shown in Figure-4.

The determination of groups was based on basic read/write operations (CRUD) in computer systems and operations specific to mobile devices (Broadcast, Control, Bind). These groups are: Access(A), Modify(M), Set(S), Update(U), Write(W), Read(R), Get(G), Manage(Mn), Bind(Bd), Broadcast(B) and Control(C). The android permissions in each group were shown in Table-7 (Appendix-1). Within the groups determined in Table-7, all features were first scanned and brought together on a group basis. The values of these features were aggregated within each group, so that the existence of the permission represents itself in the total. For example, instead of using 21 different features in model trainings separately, they were combined under the Access (A) feature. Thus, 21 properties were represented only by the Access (A) property. Although there are many permissions/features in the Android operating system, very few of them were used in applications, so it is not necessary to use all features separately during training and testing. In the Drebin dataset used in this study, the average number of permissions requested for each application is 5. Similarly, the average number of permit requests in the Genome and Arslan datasets is 6 and 5, respectively. In this case, a feature vector created with 5 numbers 1 and 344 numbers 0. Instead, all permissions were searched for each application, and they are combined under 11 feature groups after

grouping and aggregation. Thus, in the process starting as 349 features, a feature vector with only 11 features were obtained for each application.

The feature grouping process was carried out using the code structure given in Figure-5. The feature vector of 7622 x 349 dimensions taken as input was converted into a 7622 x 11 dimensional vector, thus providing a much more efficient learning in training and testing processes. The effect of the obtained vector on the results was shown in detail in Section-4.

4. Experimental Design And Results

The performance of the proposed model is evaluated in terms of i) classification success ii) time needed for training and prediction. A binary classification was made as benign and malicious. The tests were performed separately on machine learning techniques (RF, DT, LDA, etc.), DNN, RNN, LSTM and GRU networks, and the results are given in detail. Thus, it was desired to show the effect of feature grouping and reduction process used by FG-Droid application on the results.

4.1. Evaluation Metrics and Experimental Setup

The experimental environment and evaluation criteria are very important to analyze the performance of the machine-learning model. For the experiment, 70% of the data was split for training and 30% for testing. Auc, precision, recall, f-score and accuracy values were calculated to evaluate the performance of the proposed model. The calculation equations of these metrics were shown below.

$$precision = \frac{TP}{TP + FP}$$

1

$$recall = \frac{TP}{TP + FN}$$

2

$$f - score = \frac{\frac{TP}{TP+FN} * \frac{TP}{TP+FP}}{\frac{TP}{TP+FN} + \frac{TP}{TP+FP}}$$

3

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

4

TP is the number of truly malicious samples from those predicted as malware, FP is the number of samples that are predicted to be malware that are not truly malicious. FN is the number of samples that are predicted to be benign that are not truly benign and TN is the number of truly benign samples from those predicted as benign.

The computer system architecture used in the development of the FG-Droid tool is as shown in Table-1. It is very important in calculations regarding training and test times. All comparative results were obtained using the same infrastructure.

Table 1
Computer System Architecture

Parameter	Value
CPU	Intel Core i5-10760
RAM	8 Gigabyte
Operating System	Windows 10 Pro
Python version	3.7.6
Libraries	Scikit learn, matplotlib, pandas, seaborn, numpy, imblearn

4.2. Dataset

In studies on Android malware detection, it is not possible to reach sufficiently large, homogeneously distributed and reliable datasets. In this study, Drebin [44] and Genome [45] datasets were used for malware applications. The Drebin dataset contains 5560 samples from 179 different application groups. A malicious dataset with 6660 samples was created by taking 1000 samples from the genome dataset. Arslan [52] dataset was used for benign applications. In this dataset, the applications with the highest number of downloads in Google Playstore were selected in various categories, and they were subjected to security tests on virustotal.com [46] and a data set consisting of 960 applications that passed the test was created. During the labeling phase of the datasets, Drebin and Genome were distributed as labeled, so no action was taken on the malicious dataset. However, labeling in the benign dataset was made by us to be used in this study. As a result, a dataset containing real-world applications and examples that will not create noise for both malicious and benign applications has been created.

4.3. Hyper-parameter tuning for best machine learning models

In order to achieve successful classification performance in a test environment where each sample is represented by 11 features, 10 different machine learning techniques and DNN, RNN, LSTM and GRU

networks were designed and tested. At the end of these processes, hyper-parameter tuning was performed for the Extra Tree classifier, where the best results were obtained in the default parameters. Both GridSearch and RandomSearch algorithms were used for the selection of the best parameters, and the selection range of the parameters was as shown in Table-2.

Table 2
Details of the hyper-parameters of ExtraTree Classifier

Parameter Name	Hyper-parameter Tuning Range	Best Value
n estimators	200-2000	1800
Max features	Auto, sqrt	sqrt
Max depth	10-110	30
Min samples split	2,5,10	10
Min samples leaf	1,2,4	1
bootstrap	True, False	False

4.4. Results for Machine Learning

After the feature-grouping algorithm used in the development process of the FG-Droid, the results obtained in the tests using machine-learning techniques were as shown in Table-3 for 10 different classifiers.

Table 3
ML Classification Results with Proposed Feature Grouping Algorithm

ML Classifier Algorithm	AUC_score	Precision	Recall	F-score	Accuracy
KNeighbors	0.965	0.905	0.913	0.909	0.910
SVC	0.950	0.844	0.931	0.885	0.881
GradientBoosting	0.965	0.928	0.899	0.913	0.916
Random Forest	0.976	0.942	0.904	0.923	0.925
XG Boost	0.976	0.941	0.905	0.923	0.925
Extra Tree	0.979	0.944	0.909	0.926	0.929
Ada Boost	0.955	0.917	0.872	0.894	0.898
Decision Tree	0.970	0.939	0.901	0.920	0.922
Logistic Regression	0.881	0.790	0.816	0.803	0.803
Linear Discriminant	0.856	0.769	0.814	0.791	0.788

As can be seen in Table 3, the accuracy rate is 90% and above, except for two algorithms, and the highest classification rate was 92.5%. While a successful result such as 94.4% was obtained in the precision value, the recall and f-score were 92%. As a result, a very high success of 97.9% was achieved in the AUC score, which is the indicator of classification success for both classes. This shows that it is not enough for permission-based applications to remove the disruptive features in android malware detection and it is not necessary to consider each permission as a separate feature. It was possible to reach the 97.9% success level by using only 11 features of the application. The effect of the FG-Droid tool is not in feature selection, but in grouping features and ensuring that each permission is represented under the group. It has been possible to prevent the loss of the distinctiveness of the feature by selecting the feature. Thus, both the number of features were greatly reduced and the effect of many features on classification is used under the group.

4.5. Results for Deep Learning Models

During the development of the FG-Droid, tests were carried out using deep learning models. As stated before, the number of features decreases considerably with the proposed grouping algorithm. The effect of the algorithm in deep learning models, which need to use more features and more training examples in its basic structure, is very important. These tests were carried out to observe whether the group-based feature vector would have a negative effect on the classification performance in deep learning models.

Table 4
Deep Learning Models Classification Results with Proposed Feature Grouping Algorithm

Deep Learning Models	auc_score	Precision	Recall	F-score	Accuracy	Toplam İşlenen Parametre Sayısı	Epoch Sayısı
DNN(30,30)	0.918	0.936	0.906	0.921	0.925	1352	50
DNN(30,30,30)	0.920	0.927	0.916	0.921	0.921	2282	50
DNN(30,30,30,30)	0.935	0.942	0.908	0.925	0.930	3212	50
DNN(100,100,100)	0.925	0.944	0.891	0.917	0.924	31702	50
DNN(300,300,300)	0.928	0.940	0.904	0.921	0.926	275102	50
RNN(10,10)	0.900	0.910	0.910	0.920	0.902	242	50
RNN(30,30)	0.916	0.928	0.891	0.939	0.918	1322	50
RNN(100,100)	0.867	0.899	0.830	0.863	0.875	11402	50
RNN(300,300)	0.887	0.905	0.865	0.885	0.895	94202	50
GRU(10,10)	0.910	0.918	0.897	0.907	0.908	712	50
GRU(30,30,30)	0.902	0.915	0.887	0.901	0.905	3932	50
GRU(100,100)	0.915	0.836	0.891	0.913	0.913	34102	50
GRU(300,300)	0.914	0.934	0.893	0.913	0.915	282302	50
LSTM(10,10)	0.920	0.939	0.891	0.914	0.918	902	50
LSTM(30,30)	0.909	0.890	0.908	0.908	0.910	5102	50
LSTM(100,100)	0.916	0.940	0.890	0.914	0.920	45002	50
LSTM(300,300)	0.916	0.937	0.892	0.914	0.920	375002	50

As can be seen in Table-4, a lower classification success has been achieved compared to machine learning models, due to the reduction in the number of features. This decrease occurred for all metrics. The highest level of success was achieved with the LSTM (100,100) model with 92.2% for accuracy. Precision, recall and f-score values are 94.4%, 91.6% and 93.9%, respectively. The highest value of 92.5% was obtained in the AUC score, in which both classes were evaluated together. The learning curves for the models with the highest classification rate for DNN, RNN, GRU and LSTM were as shown in Figure-6. For all models, learning takes place very quickly, with training reaching its peak at 20 epochs. The test curve is parallel to the training curve. However, the learning rate slows down after 20 epochs. This slowdown was thought to be due to the need for more data to continue learning. Repeating tests with a larger dataset will allow FG-Droid to achieve a higher AUC.

4.6. Comparison Results and Best Classifier Results Details

In the FG-Droid development process, it has been understood that the models in which the proposed algorithm produces the most successful results are based on the machine learning algrithms. Machine learning techniques have generally shown high success. As a result of these tests, the most successful classification was achieved with the Extra Tree algorithm, with an AUC value of 97.7%. In order to analyze the results of this algorithm further, the ROC curve formed according to the TP and FP ratios was given in Figure-7.

As seen in Figure-7, the TP rate value increases rapidly and the area under the graph is at the level of 97.7%. The reason for using the AUC value in the comparison of classifiers is to correctly identify the model that performs well in terms of overall prediction accuracy. Thus, it will be possible for the FG-Droid tool to have the best classification performance.

In order to observe whether the proposed model performs well in both classes, the recall-precision curve was given in Figure-8. While precision is a measure of result relevance, recall shows how many truly relevant results were returned. High precision value was related to low FP rate, high recall value is related to low FP rate. As can be seen in Figure-8, the Recall value and the precision value reach the best result at a value of 90% and above. In this case, it was understood that the correct classification can be made for both the benign and the malicious classes in the classification with the proposed model.

4.7. The effect of proposed feature grouping on learning and testing time

The proposed feature grouping based algorithm performs a very large feature vector size reduction and feature selection from 349 features to 11 features. Thus, the amount of data was reduced by 30 times and a much simpler feature vector is obtained. This has a serious impact on the training and test duration as well as on the classification result. Having hardware limitations in mobile devices and the need for fast and efficient tools are another advantage in choosing FG-Droid.

In the initial state of the created feature vector, in the case of using well-known feature selection methods and in the tests made with the proposed feature grouping method, the required times for training and testing were as shown in Table-5.

Table 5
Training and testing time with different feature selection algorithms and proposed model

Feature Selection Method	Selected/Grouped Feature Size	Training time(s.)	Prediction time(s.)
FG-Droid (Proposed Model)	11	0.344	0.063
Without feature selection	349	2.234	0.109
Feature importance with extra tree algorithm	46	0.609	0.078
Feature importance with random forest algorithm	45	0.594	0.078
chi2	35	0.500	0.078
f_classiif	35	0.519	0.078
f_regression	35	0.516	0.078
PCA	35	0.688	0.078

The results obtained when using known and feature selection methods such as Extratree, randomforest, chi2, f_classif, f_regression and PCA or without feature selection are as shown in Table-5. The FG-Droid tool reaches 97.7 AUC with only 11 features. The minimum number of features required to obtain similar AUC values is 35. It has been seen that the increase in the number of selected features has a serious effect on both training and prediction time. FG-Droid is on average 700% faster in training time than in the model without any feature selection, while it is approximately 80% faster in prediction time. On the other hand, the best results were obtained with chi2 among the traditional feature selection methods, and the proposed model is 45% faster in training time and 23% faster in testing time compared to chi2 method. As a result, by grouping the features with FG-Droid, the effect of the feature on the classification is not lost and an efficient classification was made.

4.8. Comparison with similar works and discussion

A lot of work has been done in recent years on Android malware detection. These studies can basically use static analysis, dynamic analysis or hybrid feature extraction methods. While some of these obtained features contribute positively to the classification performance, some may have no effect at all, and some may have a deteriorating effect. For this reason, it is beneficial to determine those features that contribute positively to the result and to remove the others from the feature set. In this study, a feature grouping method was proposed for the use of features extracted from static analysis in classification. FG-Droid uses the feature vector consisting of grouped features and makes classification with the ExtraTree algorithm. Instead of selecting and removing features from the feature vector, the approach of evaluating these features within the group has been adopted. The Drebin malicious dataset, which has been widely

used for many years, was used in the tests of FG-Droid. The results of recent studies using the Drebin dataset and conducted in 2021 and the test results of the FG-Droid were as shown in Table-6.

Table 6
Classification Performance of Malware Detection Tools working with Drebin

Paper	Dataset	Feature selection or grouping algorithms and classification methods	Classification Performance
Mat et al. [6] (2021)	Drebin	Information gain and chi2 for feature selection, classification with bayesian.	91.10%
Arif et al. [47] (2021)	Drebin	No feature selection, classification with Risk-based Fuzzy analytical hiyerarchi	90.54%
Millar et al. [48] (2021)	Drebin	No feature selection, classification with multi-view deep learning	91.00%
Zhang et al. [49](2021)	Drebin	No feature selection. Classification with Text CNN	95.20%
Taheri et al. [42] (2021)	Drebin	Feature selection with randomforest, classifition with First nearest neighbors(FNN)	94.10%
FG-Droid	Drebin	Feature selection with proposed permission grouping algorithms, classification with ExtraTree	97.70%

Among these studies, in which the Drebin dataset was used, feature selection was made in some of them, and in others, feature vectors containing a large number of features were used for classification. The results show that the success rate of classification is between 91% and 97.7%. It is very important that FG-Droid provide similar or better performance than other works using only 11 features. This proves that FG-Droid can achieve high classification success with a fast and efficient training and testing time by processing the least data. When other studies were examined, it is understood that permission-based features are too much for application classification and it is considered necessary to choose from these features. Feature selection was carried out with methods such as information gain, chi2, randomforest. In this study, the feature selection approach was handled from a different perspective and instead of selecting features; a joint evaluation approach is adopted.

5. Conclusion

The increase in mobile devices using Android operating system has caused them to be the target of cyber attackers. New types of malware are emerging every day, and new methods were proposed as a precaution. FG-Droid uses a permission grouping-based approach to android malware analysis. It has an AUC of 97.7% with 11 features for binary classification. 349 features extracted from Android applications

are grouped and reduced to 11 features with using proposed algorithm. Thus, a much more efficient feature vector is revealed. Drebin and Genome malware datasets were used to observe the effect of the model. With the success of classification, a very efficient model has been created with the shortening of training and prediction times. In the future, tests will be performed with datasets containing more samples to further increase the classification success. In addition, since a very fast method has been developed, we aim to present it with a platform that will serve online. As a result, FG-Droid is expected to contribute positively to the security of Android smart devices.

Declarations

Confict of interest

The authors declare that they have no confict of interest.

References

[1] Palak Khanna, Amandeep Singh, “Google Android Operating System: A Review”, International Journal of Computer Applications, 174(4), 1-4, 2016.

[2] https://www.businessofapps.com/data/android-statistics/

[3] https://securelist.com/it-threat-evolution-q1-2021-mobile-statistics/102547/

[4] Xinning Wang, Chong Li, “Android malware detection through machine learning on kernel task structures”, Neurocomputing, 435, 126-150, 2021.

[5] Stephan Smalley, Robert Craig, “Security Enhanced Android Bringing Flexible MAC to Android”, Network and Disctributed System Security Sypmpsium, 1-18, 2013.

[6] Sharfah Ratibah Tuan Mat, Mohd Faizal Ab Razak, Mohd Nizam Mohmad Kahar, Juliza Mohammad Arif, Ahmat Firdaus, “A Bayesian probability model for Android malware detection”, ICT Express, 1-8, 2021.

[7] Vikas Sihang, Manu Vardhan, Ashawani Swami, Pradeep Singh, “Signature based malicious behaviour detection in android”, computer science, communication and security, 251-262, 2020.

[8] Zahoor-Ur Rahman, Sidra Nasim Khan, Khan Muhammed, Jong Weon Lee, Zhihan Lv, Sung Wook Baik, Peer Azman Shah, Khalid Wan, Irfan Mehmood, “Machine learning-assisted signature and heuristic-based detection of malwares in Android devices”, Computer & Electrical Engineering, 69, 828-841, 2018.

[9] Xin Su, Lijun Xiao, Wenjia Li, Xuchong Liu, Kuan-Ching Li, Wei Liang, “DroidPortrait: Android Malware Portrait Construction Based on Multidimensional Behavior Analysis”, Applied Sciences, 10(11), 1-20, 2020.

[10] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, Fabio Martinelli, “MADAM: Effective and Efficient Behaviour-based Android Malware Detection and Prevention”, IEEE Transactions on Dependable and Secure Computing, 15(1), 1-15, 2018.

[11] Long Chen, Chunhe Xia, Shengwei Lei, Tianbo Wang, “Detection, Traceability, and Propagation of Mobile Malware Threats”, IEEE Access, 9,1-23, 2021.

[12] Long Nguyen Vu, Souhwan Jung, “Admat: a cnn-on-matrix approach to Android malware detection and classification”, IEEE Access, 9, 1-15, 2021.

[13] Nada Lachtar, Duha Ibdah, Anys Bacha, “Toward Mobile Malware detection through convolution neural networks”, IEEE Embedded Systems Letters, 13(3), 1-4, 2021.

[14] Wikas Sihang, Manu Wardhan, Pradeep Singh, Gauraw Choudhary, Seiil Son, “De-LADY: Deep learning based Android malware detection using dynamic features”, Journal of Internet Services and Information Security(JISIS), 11(2), 34-45, 2021.

[15] Fau Ou, Jian Xu, “S^3Feature: A static sensitive subgraph-based feature for android malware detection”, Computer & Security, 112, 1-17, 2022.

[16] Francisco Handrick da Costa, Ismael Medeiros, Thales Menezes, Joao Victor da Silva, Ingrid Lorraine da Silva, Rodrigo Bonifacio, Krishna Narasimhan, Marcio Ribeiro, “Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware detection”, Journal of Systems and Software, 183, 1-14, 2020.

[17] Recep Sinan Arslan, İbrahim Alper Doğru, Necaattin Barışçı, “Permission-based malware detection system for Android using machine learning techniques”, International journal of software engineering and knowledge engineering, 29(1), 43-61, 2019.

[18] Saleh M. Shehata, Ahmed H. El Fiky, Mohammed Sh. Torky, Tamer H. Farag, Nesrin Ahmed Abbas, “Android malware prevention on Permission based”, International journal of applied engineering research, 15(1), 5-11, 2020.

[19] Janai Thiyagarajan, A. Akash, Brindha Murugan, “Improved real-time permission based malware detection and clustering approach using model independent pruning”, IET Information Security, 14(5), 531-541, 2020.

[20] Jaemin Jung, Hyunjin Kim, Dongjin Shin, Myeonggeon Lee, Hyunjae Lee, Seong-je Cho, Kyoungwon Suh, “Android Malware Detection Based on Useful API Calls and Machine Learning”, IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 175-178, 2018.

[21] Abdurrahman Pektaş, Tankut Acarman, “Deep learning for effective Android malware detection using API call graph embeddings”, Soft Computing, 24(2), 1027-1043, 2020.

[22] Akanksha Sharma, Subrat Kumar Dash, “ Mining API Calls and Permissions for Android Malware Detection”, International Conference on Cryptology and Network Security, 191-205, 2014.

[23]Moutaz Alazab, Mamoun Alazab, Andrii Shalanginov, Abdelwadood Mesleh, Albara Awajan, “Intelligent mobile malware detection using permission requests and API calls”, Future Generation Computer Systems, 107, 509-521, 2020.

[24] Vasileios Syrris, Dimitris Geneiatakis, “On machine learning effectiveness for malware detection in Android OS using statis analysis data”, Journal of information security and applications, 59, 1-22, 2021.

[25] Hossein Fereidooni, Mauro Conti, Danifeng Yao, Alessandro Sperdutti, “ANASTASIA: Android mAlware detection using Static analySIs of Applications”, 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 1-5, 2016.

[26] Yousra Aafer, Wenliang Du, Heng Yin, “DroidAPIMiner: Mining API-Level Features for robust malware detection in android”, International conference on security and privacy in communcation systems, 86-103, 2013.

[27] Dong-Jie Wu, Ching Hao Mao, Te En Wei, Hahn Ming Lee, Kuo Ping Wu, “DroidMat: Android malware detection through manifest and api calls tracing”, 7th Asia joing conference on information security, 1-8, 2012.

[28] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano de Cristofaro, Gordon Ross, Gianluca Stringhini, “MAMADROID: Detecting Android malware bu bulding markov chains of behavioral models”, Transactions on Privacy and Security, 22(2), 1-34, 2019.

[29] Asma Razgallah, Raphael Khoury, Sylvain Halle, Kobra Khanmohammadi, “A survey of malware detection in Android apps: Recommendations and perspectives for future research”, Computer science review, 39, 1-17, 2021.

[30] Lok Kwong Yan, Heng Yin, “Droidscope seemlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis”, 21st usenix conference on security symposium, 29-32, 2012.

[31] Gianluca Dini, Fabio Martinelli, Andrea Saracino, Daniele Sgandurra, “madam a multi level anomaly detector for android malware”, international conference on mathematical methods, models and architectures for computer network security, 1-17, 2012.

[32]Neha Bala, Aemun Ahmar, Wenjia Li, Fernanda Tovar, Arpit Battu, Prachi Bambarkar, “Droidenemey battling adversarial example attacks for Android malware detection”, Digital communications and networks, 1-10, 2021.

[33] Khaled Bakour, Halil Murat Ünver, “The Android malware static analysis: Techniques, Limitations and Open challenges”, 3rd international conference on computer science and engineering(UBMK), 2018.

[34] Mohammed K. Alzaylaee, Suleiman Y. Yerima, Sakir Sezer, “DL-Droid deep learning based android malware detection using real devices”, Computer and security, 89, 1-11, 2020.

[35] Michael Bierma, Eric Gustafson, Jeremy Erickson, David Fritz, Yung Ryn Choe, “Andlantis large-scale android dynamic analysis”, Workshop on Mobile Security Technologies (MoST), 1-8, 2014.

[36] Roopak Surendran, Tony Thomas, Sabu Emmanuel, “A TAN based hybrid model for android malware detection”, Journal of Information Security and Applications, 54, 1-11, 2020.

[37] Alejandro Martin, Raul Lara-Cabrera, David Camacho, “Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset”, Information Fusion, 52, 128-142, 2019.

[38] Fei Tong, Zhen Yan, “a hybrid approach for mobile malware detection in Android”, Journal of parallel and distributed computing, 103, 22-31, 2017.

[39] Li Chen, Mingwei Zhang, Chih Yuan Yang, Ravi Sahita, “Semi supervised classification for dynamic android malware detection”, Cyrtopgraphy and Security, 1-19, 2017.

[40] Guiller Suares Tangil, Santanu Dash, Mansour Ahmadi, Johannes Kinder, “Droidsieve fast and accurate classification of obfuscated android malware”, international conference on data and application security and privacy, 1-13, 2017.

[41] Hui Juan Zhu, Zhu Hong You, Ze Xuan Zhu, Wei Lei Shi, Xing Chen, Li Cheng, “droiddet effective and robust detection of android malware using static analysis along with rotation forest model”, neurocomputing, 272, 638-646, 2018.

[42] Rahim Taheri, Meysam Ghahramani, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Moura Conti, “Similarity-based android malware detection using hamming distancec of static binary features”, Future generation computer systems, 105, 230-247, 2020.

[43] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, Michalis Faloutsos, “Profiledroid multi layer profiling of android applications”, 18th annual international conferencec on mobile computing and networking, 137-148, 2012.

[44] https://www.sec.cs.tu-bs.de/~danarp/drebin/.

[45] http://www.malgenomeproject.org/.

[46] ttps://www.virustotal.com/gui/home/upload.

[47] Juliza Mohammed Arif, Mohd Faizal Ab Razak, Sharfah Ratibah Tuan Mat, Suryanti Awang, Nor Syahidatul Nadiah Ismail, Ahmad Firdaus, “ Android mobile malware detection using fuzzy AHP”, Journal of Information Security and Applications, 61, 1-11, 2021.

[48] Stuart Millar, Niall McLaughlin, Jesus Martinez del Rincon, Paul Miller, “Multiview deep learning for zero-day Android malware detection”, Journal of Information Security and Applications, 58, 1-14, 2021.

[49] Nan Zhang, Yu-an Tan, Chen Yang, Yuanzhang Li, “Deep learning feature exploration for Android malware detection”, Applied soft computing, 102, 1-7, 2021.

[50] https://developer.android.com/docs

[51] Vasileios Syrris, Dimitris Geneiatakis, “ On machine learning effectiveness for malware detection in Android OS using static analysis data”, Journal of Information Security and Applications, 59, 1-22, 2021.

[52] Recep Sinan Arslan, İbrahim Alper Doğru, Necaattin Barışçı, “Permission-Based Malware Detection System for Android Using Machine Learning Techniques”, International Journal of Software and Knowledge Engineering 29, 43-61, 2019.

Figures

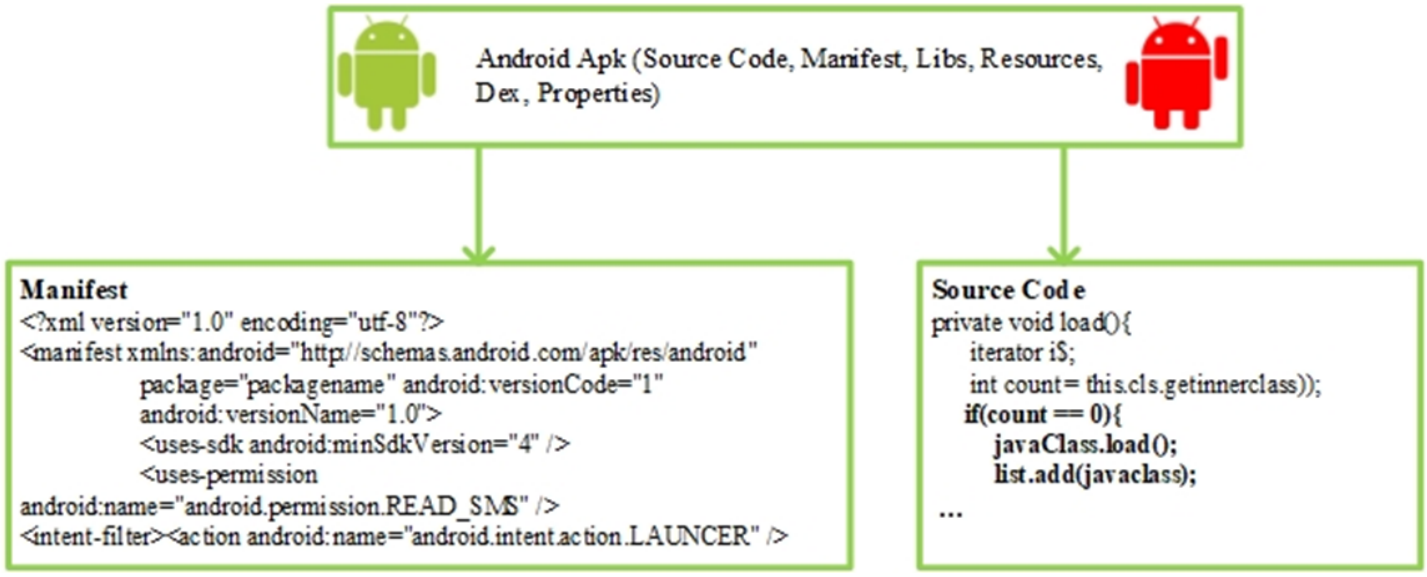

Figure 1

Structure of Android Apk [51]

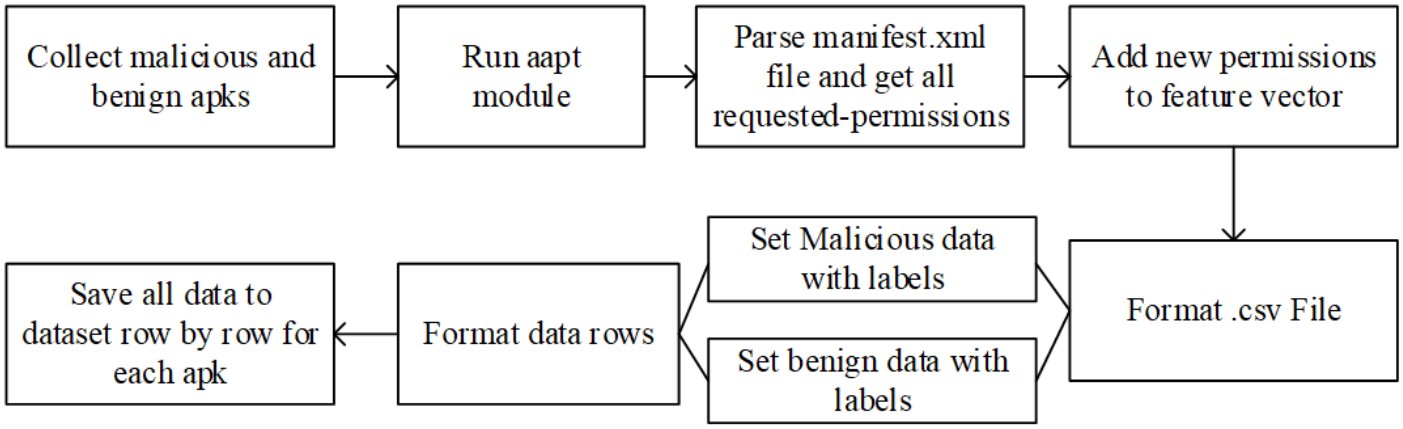

Figure 2

Automatic Feature Extraction and Pre-processing flowchart

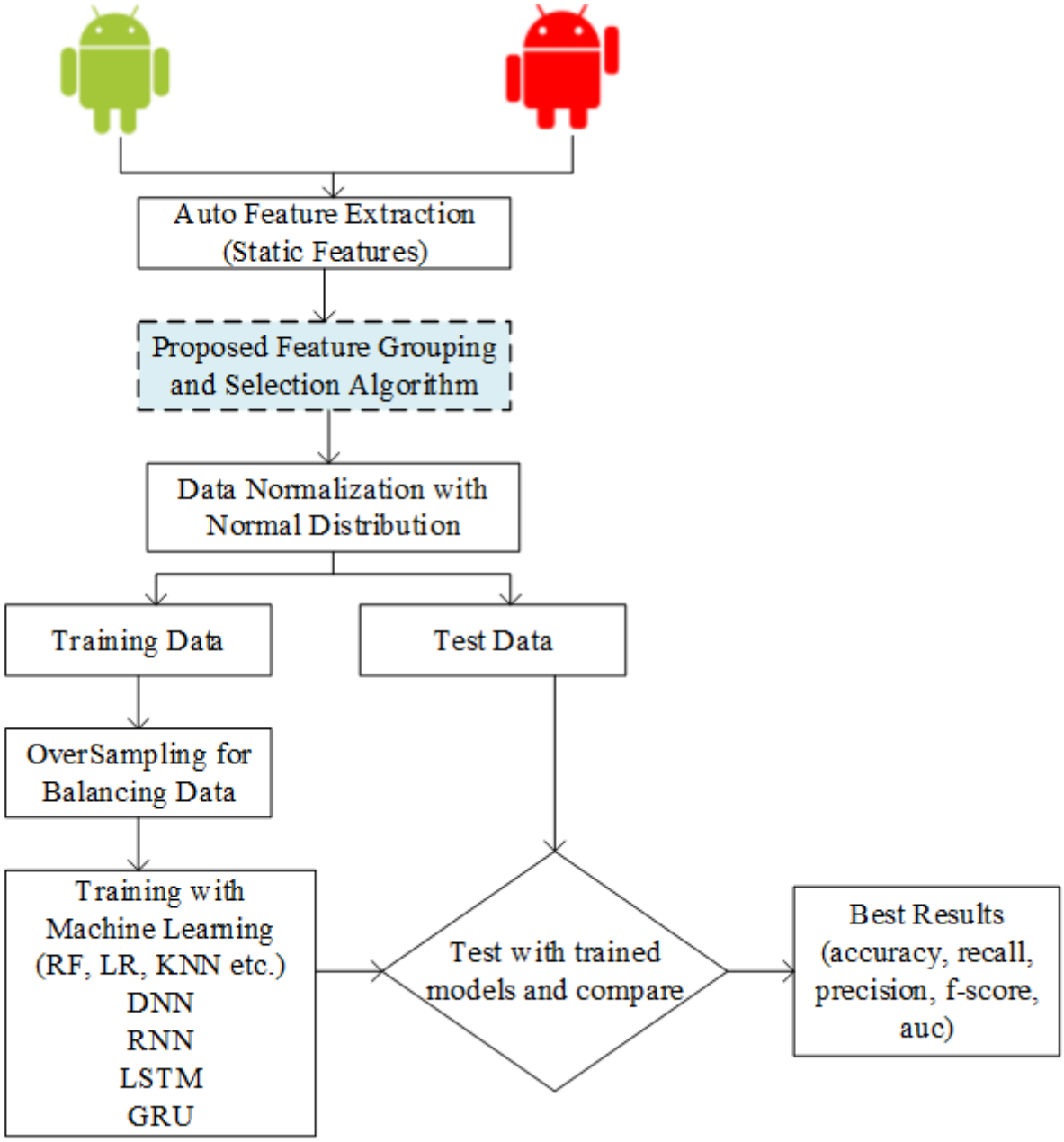

Figure 3

Flow-chart of proposed model

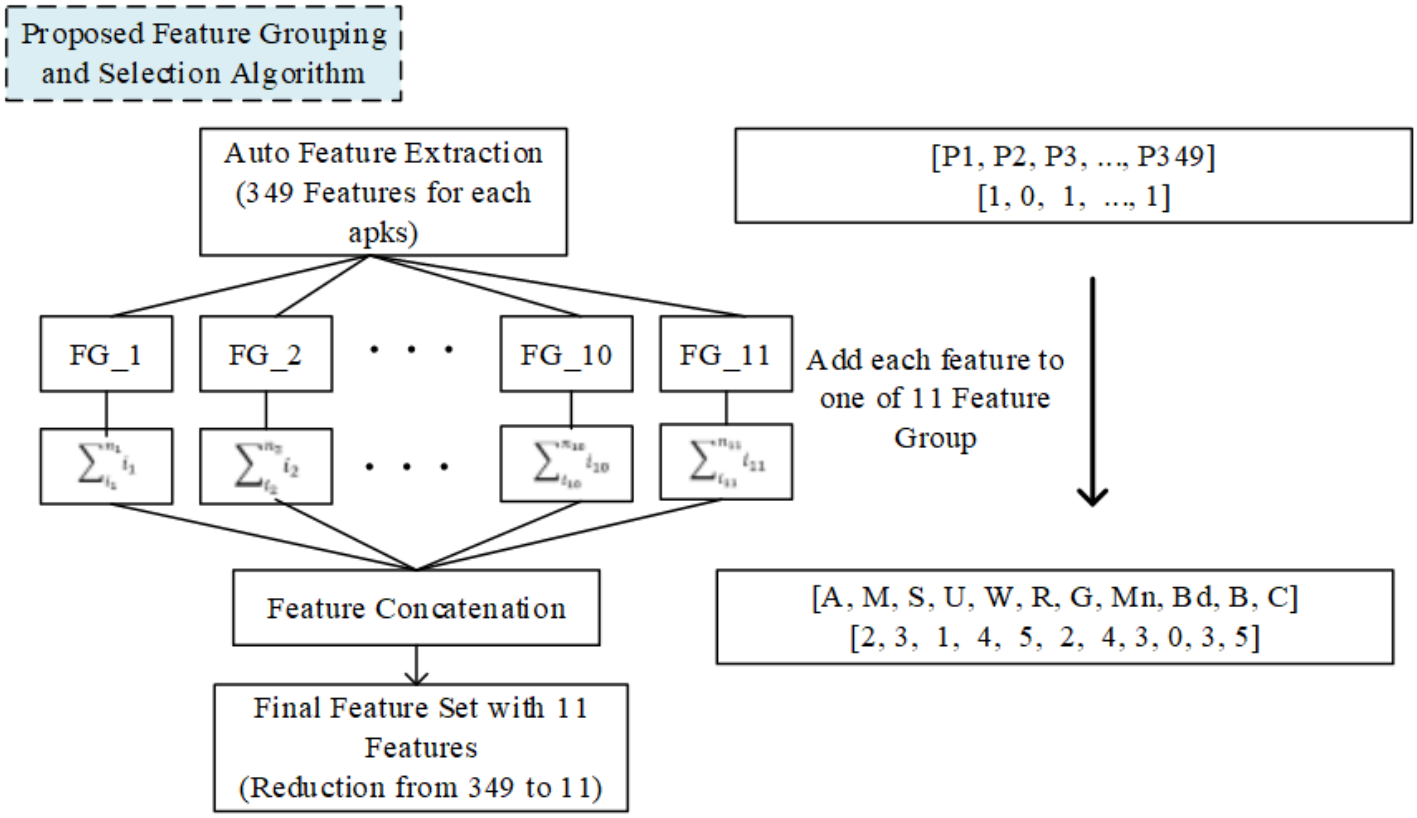

Figure 4

Proposed Feature Grouping and Selection Algorithm

```
create Final_feature_set
get all apks feature_vector (vector_size : 7622 x 349)
get Feature_groups
get default_features
loop 1 to 7622( all apks) {
     loop i in 1 to 349 {
               loop feature_index(fi) in 1 to 11{
                  if(Fi in Groupfi){
                                 Groupfi_value += Fi_value
               }
      }
        add Groupfi_value to Final_feature_set
}
return Final_feature_set (vector_size : 7622 x 11)
```

Figure 5

Pseudo-code of feature grouping and selection algorithm

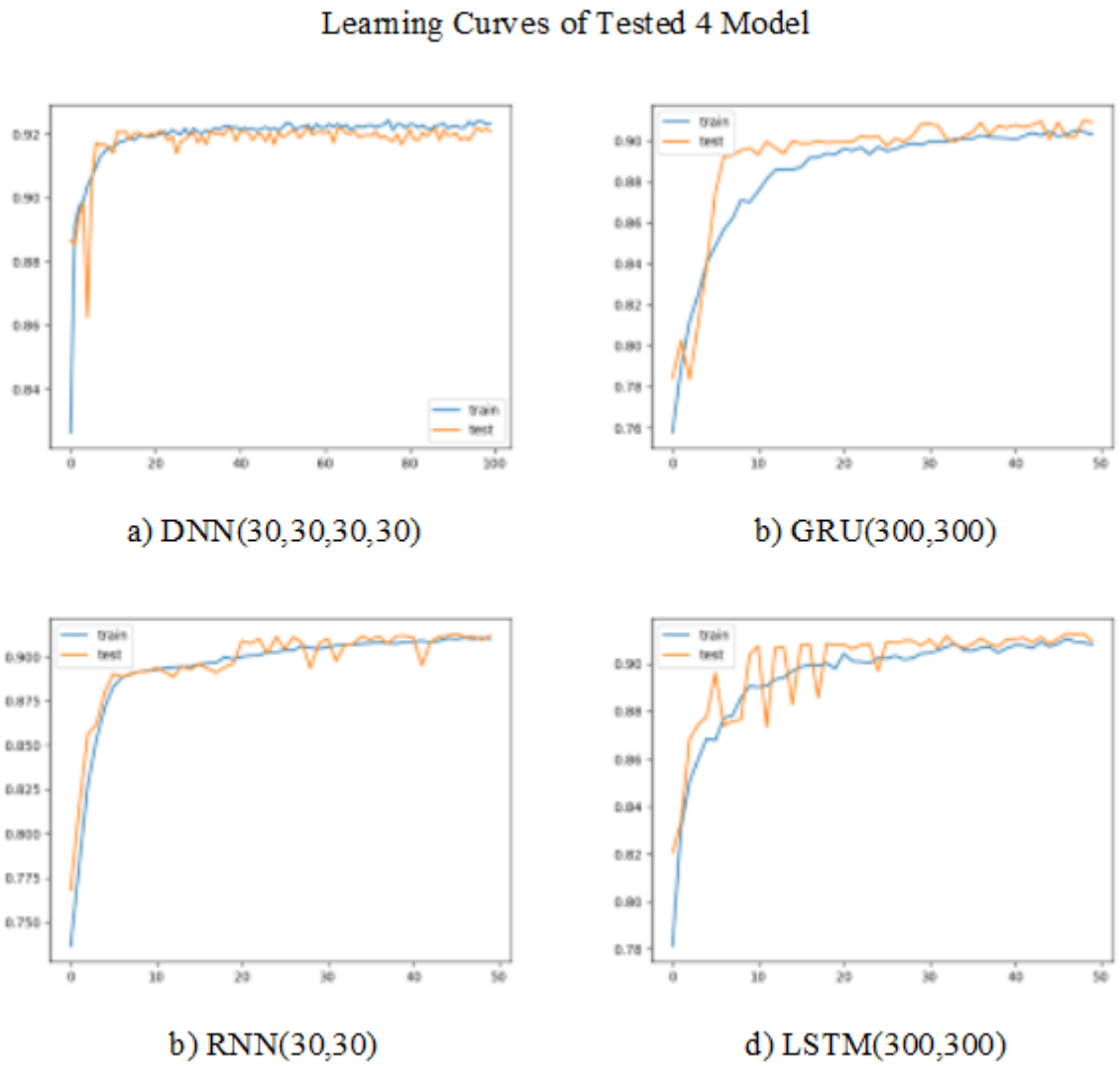

Figure 6

Learning Curves of Deep Learning Models

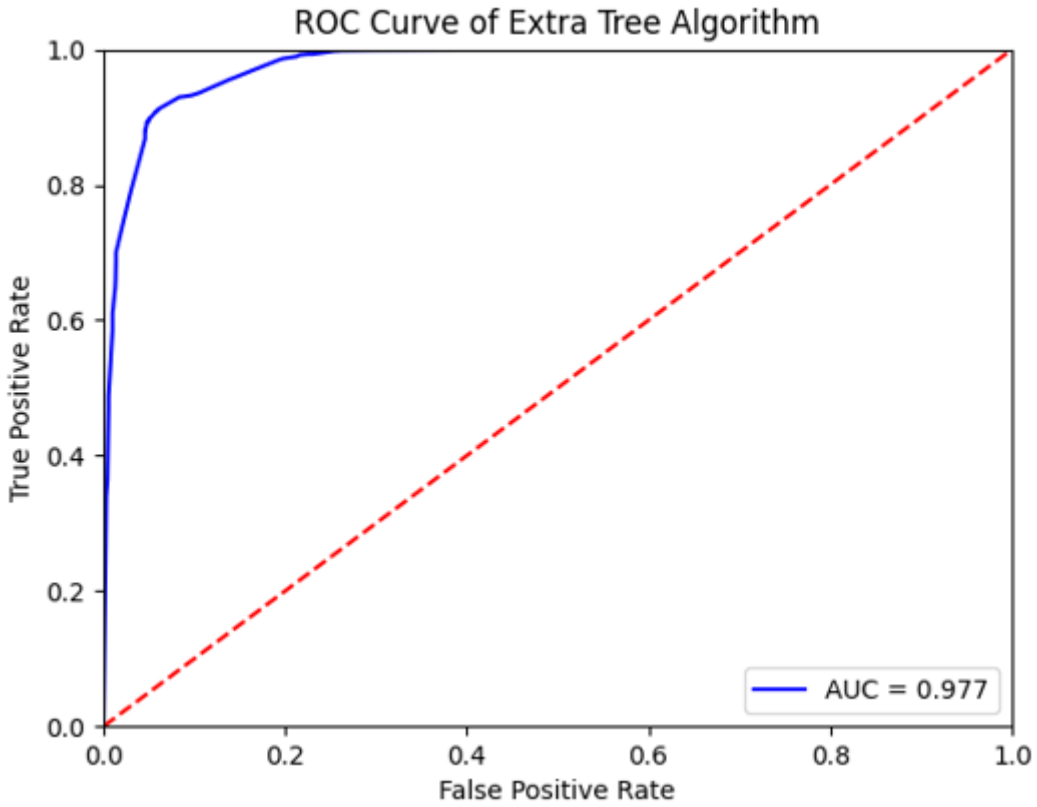

Figure 7

ROC Curve of Extra Tree Algorithm with Feature Grouping

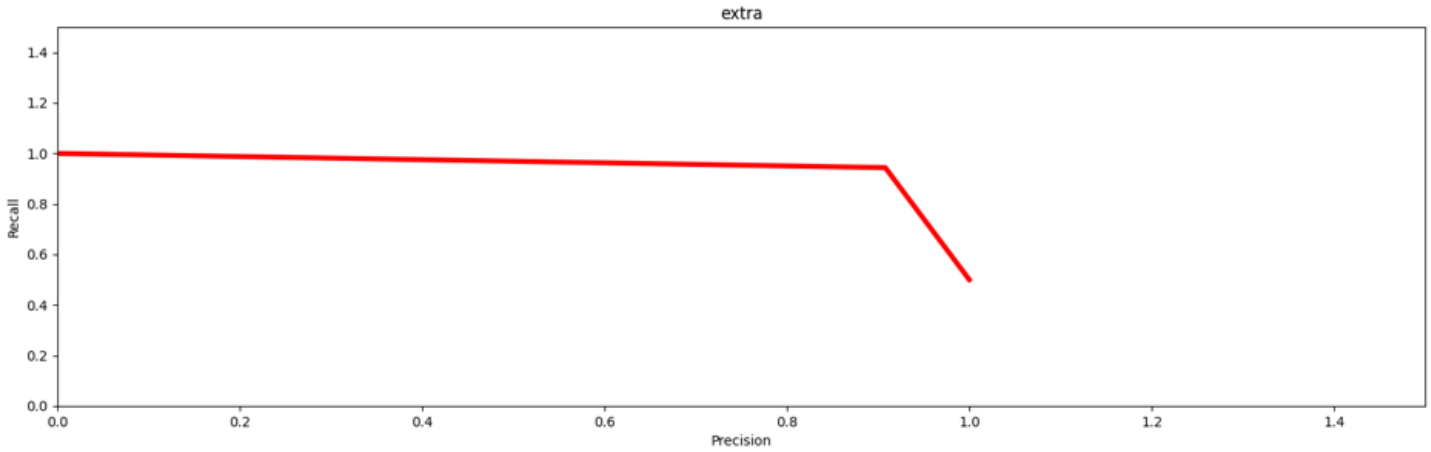

Figure 8

Precision-recall curve of Extra Tree Algorithm with Feature Grouping

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Appendix.docx