# Android Malware Detection using Machine Learning and Deep Learning Models

Varsha Meghana Kanmuri
UNI: vk2497
Email: vk2497@columbia.edu

## Abstract

This paper reviews the topic of **Android Malware Detection using Machine Learning and Deep Learning Models.** Advances in machine learning and AI have led to improved detection of malicious apps in the Android operating system.

Our paper performs a literature survey covering the latest advancements in detecting Android Malware using state-of-the-art Machine learning models. We also cover the newest research in Machine Learning in Security, Android Malware Detection techniques, and Explainability in AI. The papers also include some limitations and possible research directions to improve these Machine Learning models in the future.

As part of the experimental study, we have built machine learning models such as K-Nearest Neighbors (KNN), Decision Trees, Random Forests, Ensemble Methods, Support Vector Machines, and Neural Network Models. We use SVM and Random Forests as it has advantages in solving high-dimensional, non-linear problems. We also use Ensemble Methods and Neural Networks as they are robust to outliers (lower fault tolerance). We compare different ML models to evaluate performance and find that Neural Networks have the best accuracy and F1 score on test data. Finally, we learned through SHAP explainability and feature importance plots that android applications that send or read sms, access device location and use hidden '.exe' files during run time are typically classified as Malware with a high-level of confidence successfully by our final Neural Network classifier.

## 1. Introduction

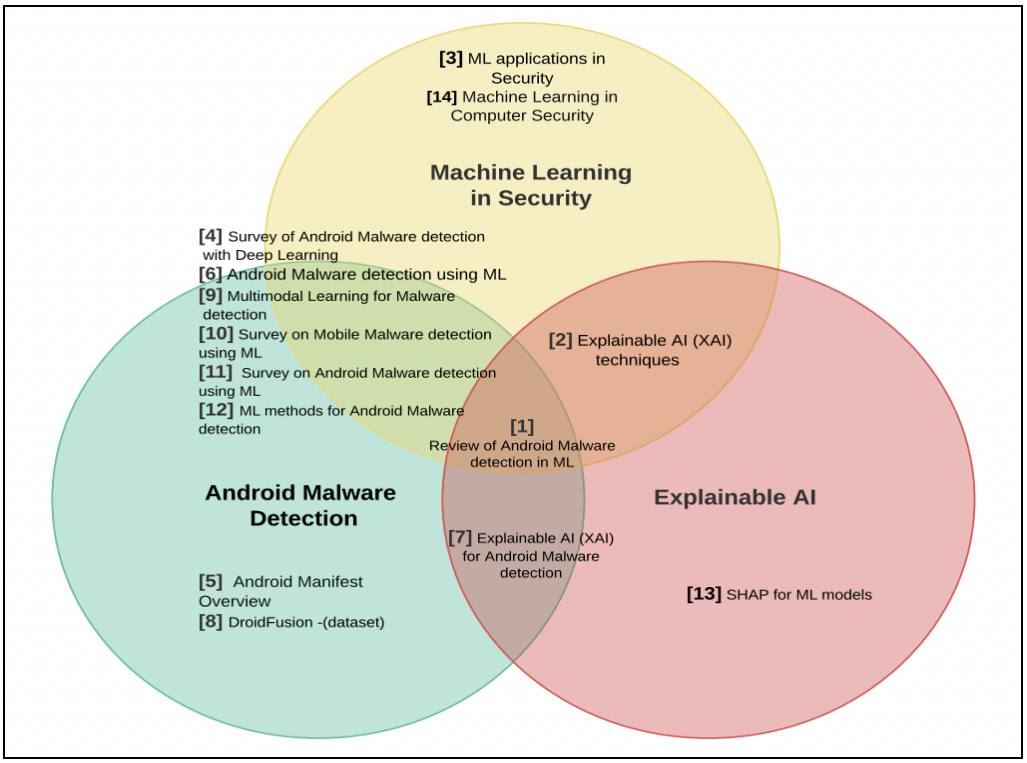
Android is one of the most commonly used operating systems in our smartphones and tablets. Google play store, one of the biggest markets for free and paid android apps, consists of over 2.6 million apps today. The widespread adoption of Android devices has made them a prime target for attackers seeking to infect large numbers of devices. The open-source nature of the Android operating system and its app distribution model makes it easier for malware to spread and more challenging for individuals and organizations to protect against. Malware detection is a very important research topic in Computer Security. Android malware detection is identifying malicious software designed to target Android devices.

Machine Learning is a branch of Artificial Intelligence that consists of a broad range of algorithms on data from various domains across the world to build models which learn to predict

and simulate the behavior of human beings. With advances in the field of Machine Learning, there have been many novel approaches to detect Android Malware using state-of-the-art Machine learning models.

Recent research has suggested that Machine Learning (ML) Models are becoming increasingly popular in Android Malware Detection. Most of the research papers show that these models perform exceptionally well in the testing phase, with a majority of them having an accuracy of above 95%, some even exceeding 99%. This gives the impression that the most promising models have already been developed and gives the impression that there is little need for further research. Unfortunately, some studies have shown that these models fail to perform the same way in the real world. Paper [2] talks about the need to utilize Explainability techniques to understand why the models perform well when trained and what they learn during training. The paper further discusses how the ML models learned to classify an app using temporal changes instead of actual characteristics which are malicious.

**1.1 Venn Diagram**: Our paper reviews papers from the three domains - Android Malware Detection, Machine Learning in Security and Explainability in Artificial Intelligence as represented by the Venn Diagram below.



### 1.2 What reader is expected to learn from the paper?

The reader is expected to learn about the state-of-the-art approaches for android malware detection using machine learning. The paper provides an overview on techniques to extract features from android applications, along with machine learning algorithms used for classification. Additionally, this paper highlights key aspects of related work from the research community - listed in the Venn Diagram above.
We answer the following research questions through many experiments of several ML models: -
RQ 1) Can ML models successfully be used for android malware detection ?
RQ 2) How different ML / Deep Learning models compare in performance ?
RQ 3) Can we explain why the ML models classified an android application as malware ?

## 2. Background

### 2.1 Machine Learning

Machine Learning models have broadly been classified into three main categories:
1) Supervised Learning - This branch of machine learning uses labeled data to make predictions about an output label. The most prevalent problems in this category include classification and regression problems.
2) Unsupervised Learning - This branch of machine learning mainly deals with datasets having unlabeled data. The models under this category try to learn and build groups or patterns by clustering input data with similar characteristics to make predictions.
3) Reinforcement Learning - In this type of Machine Learning, the models learn from their own experiences. The errors made by the model are fed back to it by a continuous feedback loop which helps the model make better decisions in future predictions.

Our paper mainly focuses on using Supervised Machine Learning models for our task of Android Malware Detection. The machine learning models which we will be primarily using in this paper for our experimental study are listed below-

**2.1.1 K-Nearest Neighbor (KNN)**: The K-Nearest Neighbor algorithm is based on the assumption that data points which are more proximate to each other are similar and belong to the same category. For a classification problem like android malware detection in our paper, the idea is to find 'k' neighboring points closest to the unknown sample to predict its class. Different distance metrics can be used to measure the distances, with Euclidean distance being the most frequently used one.

**2.1.2 Logistic regression**: We used logistic regression to classify an input sample comprising the android app's information as benign or malware. A sigmoid function maps the values predicted by the model to corresponding probabilities between 0 and 1, correlated to the two output classes, benign and malware, in our case.

**2.1.3 Support Vector Machines:** Models trained by Support Vector Machines aim to determine a hyperplane in an N-dimensional space, which optimally classifies the data points. While many hyperplanes can be found to separate the data samples, SVMs focus on finding the most optimal hyperplane, which separates the classes with maximum margin distance while maintaining its accuracy. Kernel SVM, like RBF (radial basis function) SVM, has become increasingly famous in recent years, enabling the SVM models to fit non-linear data with higher dimensions.

**2.1.4 Decision Trees:** A decision tree is one of the most interpretable supervised ML models replicating a tree-like structure, where at each level, a True/False decision is made, with typically the goal being for the leaf nodes to reach purity (all samples belonging to one class).

**2.1.5 Ensemble Methods:** Ensemble methods in Machine Learning use the results from combining multiple ML models instead of just one. The main idea behind them is that when many ML/DL models are combined, the results will be more accurate, and thus performance is improved. The main benefit of using these methods is that they can be applied to several ML algorithms like neural networks and decision trees. While these models might require additional resources like training time and memory, they have been proven to perform exceptionally well for some problems like the 'Netflix Prize' competition. They work well to reduce the models' overfitting and improve their generality. We implement the following models - Random Forests, HistGBM, XGBoost which are the state-of-the-art approaches used in the research community today. An advantage of XGBoost is that it has faster training time during cross-validation and hyper-parameter tuning compared to HistGBM; But both the models use Boosting to improve performance.

**2.1.6 Neural Networks:** Neural Networks are machine learning algorithms that draw inspiration from the structure of a human brain. They are made up of a series of interconnected nodes. A neural network generally has three parts: an input layer where each node represents the input feature values, one or more hidden layers, and an output layer where each node represents the target value(s). The nodes' connections have weights that the model learns during training. The network calculates each of these weights to minimize the difference between predicted and true outputs.
Neural networks are increasingly gaining popularity in various tasks, like image/speech recognition, computer vision, and natural language processing.A perceptron is the simplest form of artificial neural network. It is used for binary classification problems and consists only of a single layer of neurons fully connected to the input. Therefore, it can only be modeled on linearly separable data. Multi-layer perceptrons and

**2.2 SHAP (SHapley Additive exPlanation)**
SHAP [13] is a technique adapted from game theory, which can be used to explain the predictions or results of AI/ML models.

SHAP explains the prediction for an ML model's specific instance/output by considering the marginal contributions of the average of all possible permutations/coalitions of feature values.
The SHAP values can provide essential details about the model's output, like Feature Importance, Correlation between the features themselves as well as the target value, and the impact of the values of a data point/feature on the predicted output.
The SHAP values can be used for

1. Global interpretability — the collective SHAP values show the extent to which features positively or negatively influence the target variable.
2. Local interpretability — SHAP analysis is carried out on each input instance. This helps with a more detailed overview and transparency of the learning process. It answers questions like - How a minute modification in the input would modify the output for that data point.

## 3. Overview of methods in state-of-the-art research

### 3.1 Android Security Management using Android Manifest File

Android OS applications are only provided with a limited set of default permissions. These authorizations are usually specified in the AndroidManifest.xml file. This file comprises information such as the content providers, services, and a diverse set of permissions.
Android developers have made it necessary that any app which would like to access any user-specific data (sending SMS, Emails, or accessing the user's contact details) or accessing other phone/tablet-specific features like the user's local files or camera have to be specified in this file.
For example: A unique label identifies each permission. For example, sending an SMS can be defined as follows in the <uses-permission> section of the AndroidManifest.xml file as follows:

```
<manifest ... >
   <uses-permission android:name="android.permission.SEND_SMS"/>    ...
</manifest>
```

All the recent versions of the Android apps make it mandatory for the downloaded apps to first ask for access or authorization permissions from the user during runtime.

As stated in [5], Android Malware can be classified into the following categories: worm, botnet, trojan, worm, spyware, ransomware, and aggressive adware.

### 3.2 Android Malware Definition and Classification

Android Malware is malicious software or code that targets Android devices and usually aims to destroy or access the user's sensitive data without permission.

As stated in [1], Android Malware can be classified into the following categories: worm, botnet, trojan, worm, spyware, ransomware, and aggressive adware.
Today's most common malware tries to access a user's bank details or log-in credentials, access their contacts and photos, expose their data to hackers, or generate malicious advertisements.

Paper [1] mentions that most research papers view malware from three different perspectives:
Attack goals and behavior describe the intention behind a malware attack, like sensitive data theft, spamming, and financial abuse. Distribution and infection routes represent malware distributed through SMS, the user's network, or phishing emails. Privilege Acquisition Modes include user manipulation or controlling their devices.

The increasing number of financial transactions and sensitive data being managed on Android devices makes it critical to have effective malware detection and response measures in place to protect against theft and unauthorized access.
Therefore, studying Android malware detection can help individuals and organizations better understand the threats posed by malware and how to protect themselves and their assets.

This can include learning about the different types of malware that target Android devices, how they spread and infect devices, and what can be done to detect and remove them. Studying Android malware detection can also provide opportunities to develop new and innovative techniques and tools to protect against these threats.

Malware detection is among the prominent topics which attracted many researchers. Recent studies claim that there has been significant growth in different forms of malware, like worms, viruses, and trojans, attacking millions of applications worldwide.

**3.3 Different approaches to detect Malware:**
Three main approaches were identified which could be used to detect malware in the applications: signature-based detection, anomaly-based detection, and heuristics-based detection.
The static analysis, which employs the signature-based approach, compares the information scanned from the application files and compares the data with predefined signatures of the malware/virus that have already been discovered. The source code files are studied and mapped to code sequences and commonly seen patterns that belong to a specific malware variant.
Several detection techniques, such as disassembled code and annotated context-free graphs, have been intensively studied under static analysis. While static analysis has been proven to detect malware in static code or binary files, it fails to perform the same way to detect malware when the application is running. It further fails to detect malware; it fails to detect malware that is more evolved and modifies its behavior to evade detection.

The second approach, anomaly-based detection, uses a dynamic analysis strategy to detect malware when the application is up and running. Commonly used techniques use audit logs, dynamic executable files, and function calls. While this approach helps catch malware once the application code is executed, detecting "zero-day" malware attacks (an attack on a software application with a vulnerability/defect that the software manufacturer or developer is unfamiliar with) is more challenging.

Over the last few years, Heuristic-based detection techniques have gained immense popularity in malware detection. The Heuristic-based detection approach has proved to be more efficient in addressing the drawbacks mentioned in the signature-based and anomaly-based detection approaches. This approach utilizes the capability of various
Machine Learning, Deep Learning, and Data mining techniques for malware detection. The most common algorithms majorly use one of the following two techniques - 1) clustering various applications based on their behavior or 2) classifying the applications based on their behaviors as benign or malware.

Paper [1] presents a comprehensive overview of machine-learning practices in android malware detection. The authors compare the performance of various Machine Learning and Deep Learning Models used in past research, describe the benefits of using one machine learning model over the other based on their pros and cons, and suggest insights for new research pursuits. The authors shortlisted top papers from well-known conferences and reliable data sources like Google Scholar, Academia, and ResearchGate. Since research on Android Malware detection has grown exponentially recently, the authors mainly prioritized the papers published within the last five years. They used keywords like "machine learning" with "android malware detection" while collecting the documents.

**3.4 Machine Learning Process for Android Malware Detection:**

Machine Learning models have gained significant popularity in the past decade in addressing a wide range of security concerns in numerous web and android applications.
Paper [3], [4], [6],[10] discusses how Machine Learning and Deep Learning models have played a substantial role in a broad spectrum of security applications like Intrusion and Phishing Detection, Privacy Preservation, and Android Malware Detection. The paper emphasizes the need to compare and assess these models' performance and vulnerabilities to protect them from potential adversarial attacks. Furthermore, it aims to provide a framework to promote future interdisciplinary research to solve more intricate problems based on the results obtained.

The authors [11], [12] selects over a hundred security-related papers where machine learning models where applicable and further categorizes them using a comprehensive set of variables like the type of the security application, the algorithm used, year of publication, performance, and adversarial analysis. The results of the study showed that there had been a considerable boost

in the research work and the papers published in the field of machine learning for security and privacy.

The Machine Learning process, which is typically used for Android Malware detection, is as follows:
1) **Identification of the type of ML problem**: Since our paper focuses on Android Malware detection and aims at classifying whether our app is malware or a benign one, we will be building classification models for our problem.
2) **Data collection**: The next stage would be to collect samples that are a good representation of real-world apps from reliable sources like the Google play store.
Some famous datasets currently in use for research and experimentation for our problem include MalGenome belonging to the Android Malware Genome Project. Further, the Drebin project is a well-known dataset used in multiple research papers. The Drebin dataset comprises over 5,500 applications belonging to over 170 Malware families.
We will also use a sample of this dataset for our experimental study.
3) **Data processing**: This process takes care of preprocessing steps like data cleaning for missing/null values, managing class imbalances, overcoming data redundancy, and data normalization. (Table in Appendices)
4) **Feature selection**: Feature selection goes hand-in-hand with data cleaning. Feature selection is essential to eliminate features that are not very significant in the performance of our ML models. In addition, it helps overcome the "curse of dimensionality" by reducing the feature space.

**Feature types:** The features machine learning practitioners select for Android Malware detection can be broadly classified into static, dynamic, and hybrid features. Static features are selected from the source code without running the code. Files such as the AndroidManifest.xml files are excellent sources for extracting features like app permissions allowed by the users and device information.
Dynamic features are the features that can be extracted when the application is running; these may include network traffic and API calls.
Hybrid features use both static and dynamic features and are used by most machine learning researchers these days.

The following table below compares different ML algorithms which are commonly used for Malware detection.

| Model or Algorithm | Advantage | Disadvantage |
|---|---|---|
| Decision Trees (DT) | Simple to understand and interpret. It can handle samples with missing values or large scale. | Prone to lead to overfitting. It does not support online learning. |
| Naive Bayesian (NB) | The model can be trained easily and quickly. | Not applicable to situations where the feature variables are correlated. The prior probability needs to be calculated. |
| Linear Model (LM) | It is the main algorithm in statistics; fast and direct. | The premise of the algorithm is strict. It cannot deal with the high-dimensional features well. |
| Support Vector Machine (SVM) | It has advantages in solving small-scale, high-dimensional or non-linear problems. | The overhead in data processing is large. It is sensitive to samples with missing values. |
| K-Nearest Neighbors (KNN) | Easy to realize without parameter estimation. It is suitable for solving multi-classification problems. | Greatly affected by data skew. The computation overhead is relatively large. |
| K-means | Simple, fast, and easy to implement. | Results are affected by the initial setting. It is sensitive to noise and outliers. |
| Neural Network and Deep Learning (NN&DL) | Has high accuracy and strong fault tolerance. | Needs a lot of data for training. Parameter and network topology selection is not easy. |
| Ensemble Learning (EL) | Much more accurate than using a single model. | Overhead is large. It requires a lot of model training and maintenance. |
| Online Learning (OL) | Strong adaptability and good real-time performance. It reduces the threshold requirement of hardware performance. | Some models are not suitable for online learning. It is difficult to find the global optimal solution. |

Figure: - Comparison of machine learning algorithms commonly used for Android Malware Detection.

Reference - [1], Liu et. al., A Review of Android Malware Detection Approaches Based on Machine Learning

### 3.5 Why is explainability becoming increasingly important in ML models ?

Papers [2], [14] conduct experiments with some of the most prominent datasets for Android Malware Detection from the AndroZoo project ( which comprises over 11 million Android applications from 18 sources, including the official GooglePlay store and other countries like Chinese markets) and the DREBIN project.

The high accuracy in the ML models might be due to many forms of experimental biases [2]. For instance, some ML models classify the apps based on certain deprecated features irrelevant to malicious/benign behaviors on the DREBIN dataset.

Paper [14] conducted experiments on the AndroZoo project dataset and discovered the data used for the ML process had sampling bias. For example, when picking a sample without constraints on the number of detections, the probability of selecting apps from Google Play was approximately 80 %. However, for a threshold of picking ten defective apps, the probability that an app was randomly selected from the Chinese market is 70 %. The experimenters also found

that the models were prone to developing spurious correlations, as stated in [14], "Artifacts unrelated to the security problem create shortcut patterns for separating classes. Consequently, the learning model adapts to these artifacts instead of solving the actual task." The results from explainability showed that trained ML models started using the URL "play.google.com" (origins of the app) as one of the most important features for predicting the app to belong to the benign class instead of focusing on the app's actual behavior.

Additionally, I also learn from [7] where the authors developed a light-weight explainable AI system that had more than 98% accuracy, yet could be easily explained and deployed in mobile devices. The feature importances could be used to better design and implement security and anti-virus software within the android ecosystem.

These papers highlight the emerging need for Explainable AI ( XAI techniques ) to help researchers better understand how AI/ML models for android malware detection work instead of depending entirely on the evaluation metrics like accuracy, precision, or F1 scores.

## 4. Experimental Results and Discussion

We conduct extensive experiments of several machine learning algorithms to classify whether an android app is malicious or benign. These experiments help us compare the performance of different Machine Learning algorithms, which is crucial in deciding which model to deploy for real-time android malware detection.

### 4.1 Data

In this paper, we use the widely known and publicly available DREBIN-215 dataset [8] consisting of 215 features from 15,036 app samples. In this dataset, there are 9476 benign samples, and 5560 malware samples which were obtained from the DREBIN project that is well-known in the research community.

**4.2 Feature Extraction: -** The features are extracted using an automated static analysis tool in Python as shown in the figure below.

The tool is used to extract permissions and intents from the application manifest file [5]. API calls related command strings are extracted through reverse engineering from ‘.dex’ files. Finally, the static analyzer also checks for external/dangerous ‘.exe’ files and hidden linux commands within the application. Prior works [1], [8], [10] have shown successful use of such static application attributes as discriminative features for machine learning models to detect android malware. The features are represented in binary form, and each data point labeled as ‘S’ - malicious or ‘B’ - Benign.

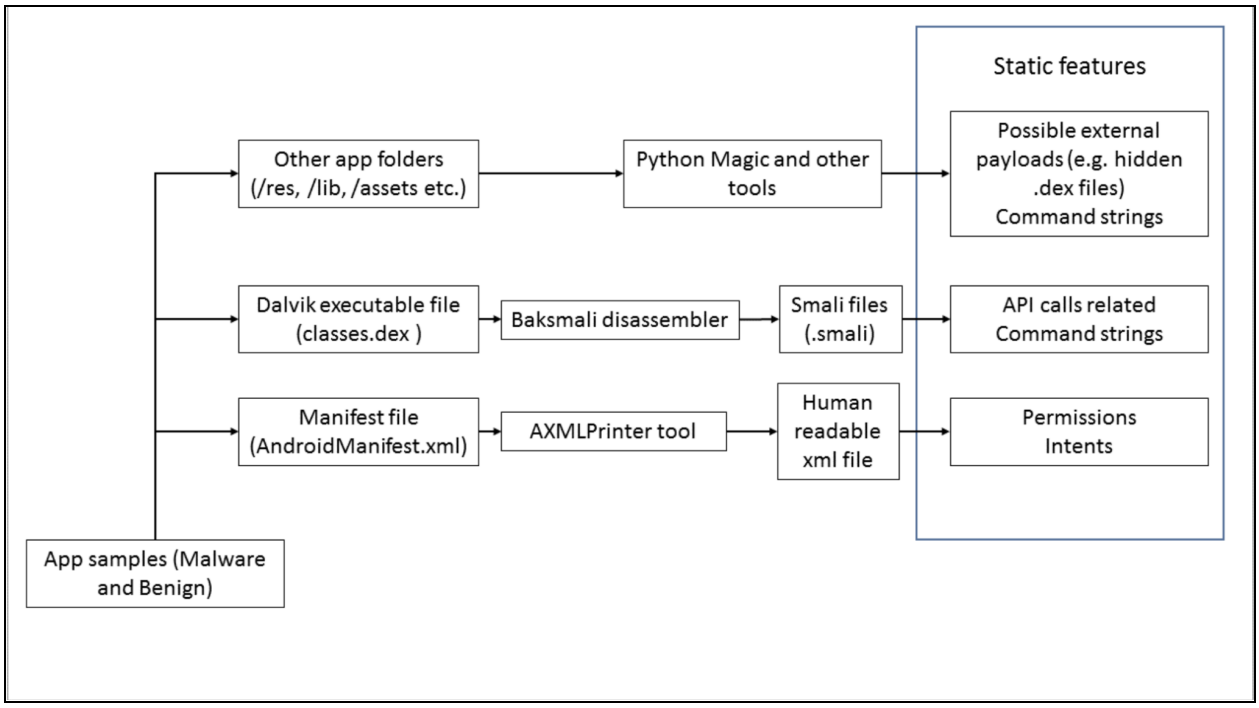



Figure: - Architecture overview of python based static analyzer for automated feature extraction from app samples.
Reference - [8], Yerima et. al., DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection

**4.3 Feature selection and Data Pre-processing: -** We pre-process the data, replacing missing values / '?' and performing label encoding which converts 'B' benign samples with label 0, and 'S' malware samples with label 1. After preprocessing the data, we remove features that are highly correlated using a threshold of 0.9 for correlation. [4], [6], [8], [11], also suggest using techniques like Information Gain (IG), Principal Component Analysis (PCA), and t-SNE, for feature ranking and selection. We split the data into train, test and validation, using 'stratified' splitting instead of random splitting as the data is imbalanced [14]. This ensures the proportion of data labels during 'train' and 'test' are equal, so we account for class imbalance while evaluating machine learning models.

**4.4 Machine Learning and Deep Learning Models**

We use 80% of the data for training with 10-fold cross validation (10% of training data for validation in each cross-validation fold) for hyper-parameter tuning in each model. We then evaluate model test performance on the remaining 20% test data.

In the experiments below, we use the following models with the fine-tuned hyper-parameters. To establish a baseline as in [1], [10], we use K-Nearest Neighbors (KNN), tuning the value of K

across a range of values (e.g. K=1 to 20) , and logistic regression classifier tuning hyper-parameters like regularization coefficients (e.g. C=0.01 to 100), learning_rate, max_iterations, etc. using 10-fold cross validation. We show the best results below for K=3,5,7 for KNN and C=10 for logistic regression. Next, we evaluate the performance of Support Vector Machines (SVM) [6], [10], [12] with linear and RBF Kernels, tuning hyper-parameters like regularization coefficients (e.g. C=0.01 to 100). We show the best performing classifier results below for the final-tuned hyper-parameters are C=5 for linear SVM, and C=10 for SVM with RBF kernel.

Although the SVM with RBF kernel performs well, we also want better explainability into key features which causes the model to classify an app as ‘malware’. So, we also evaluate performance with Decision Trees and ensemble methods like Random Forests, HistGBM, XGBoost [1], [9], [11]. We tuned the following hyper-parameters for decision trees - ‘max_depth’, ‘min_samples_split’, ‘min_samples_leaf’, ‘min_impurity_decrease’, loss_function, ‘max_leaf_nodes’ and show the best result for hyper-parameters {'max_depth': 24, 'min_impurity_decrease': 0, 'min_samples_leaf': 1, 'min_samples_split': 2}. Similarly for ensemble methods, we tune hyper-parameters such as - ‘n_estimators’, ‘max_depth’, ‘max_iter’, ‘learning_rate’, ‘gamma’, ‘lamda’ (regularization coefficient) etc. As the number and space of hyper-parameters is very large, we only mention a few best hyper-parameters for the ensemble models - Random Forests {'max_depth': 25, 'n_estimators': 400, 'criterion': 'gini', } ; HistGBM {'learning_rate': 0.1, 'max_depth': 15, 'max_iter': 100} ; XGBoost {'gamma': 0, 'lambda': 1, 'max_depth': 10}. Although Decision Trees do not perform as well as SVM with RBF Kernel ( like due to higher overfitting ), the ensemble methods have good performance comparable to SVM (RBF kernel) along with providing better explainability [2], [7]. The feature importance plots are shown in the next section.

Finally, we also use Deep Learning ( Neural Networks with different architectures ) techniques to evaluate performance, given the wide successful adoption and results for various applications including android malware detection [4]. We show the results below after fine-tuning hyper-parameters like ‘activation_function’, ‘number_hidden_layers’, ‘number_hidden_units’, ‘learning_rate’, ‘epochs’, ‘solver’, ‘regularization coefficients’, etc. The best F-1 score was obtained for the Neural Network with 3 hidden layers, and ‘relu’ activation function - {solver='adam', alpha=0.01, activation='relu', hidden_layer_sizes=(128, 64, 32)}. Although Neural Networks are very powerful and give the best results, it is very hard to interpret them and the weights of the hidden layers. So, we use SHAP to interpret the model and extract feature importance [7], [13] in the form summary plots shown later below in the ‘Explainability of Models and Feature Importance’ section.

As mentioned in the Appendix section, we use the F-1 score (weighted) metric which takes into account the class-imbalance [14] when comparing the performance for the different models.

### 4.5 Results

In this subsection, we present the classification results evaluated using different machine learning and deep learning models. I trained the models on an Apple Macbook Pro laptop with 32 GB RAM and M1 chip processor. The overall computation time is around 164 hours ( ~ 7 days ) for the experiments including setup time for writing code, plotting the SHAP explainability plots, and 10 fold cross-validation with hyper-parameter tuning for all the machine learning models. The most expensive computation time is for GridSearchCV, when tuning the hyper-parameters over a large search space.

| Machine Learning model | Accuracy Train data | Accuracy Test data | Precision Malware | Recall Malware | Precision Benign | Recall Benign | F-1 score (weighted) |
|---|---|---|---|---|---|---|---|
| K-NN (3 neighbors) | 99.18 % | 98.07 % | 0.98 | 0.97 | 0.98 | 0.99 | 0.9807 |
| K-NN (5) | 98.68 % | 98.03 % | 0.98 | 0.97 | 0.98 | 0.99 | 0.9803 |
| K-NN (7) | 98.47 % | 97.93 % | 0.98 | 0.96 | 0.98 | 0.99 | 0.9803 |
| Logistic regression Classification | 98.42 % | 97.70 % | 0.97 | 0.96 | 0.98 | 0.98 | 0.9770 |
| Linear SVM | 98.32 % | 97.83 % | 0.98 | 0.96 | 0.98 | 0.99 | 0.9783 |
| RBF SVM | 99.69 % | 98.86 % | 0.99 | 0.98 | 0.99 | 0.99 | 0.9886 |
| Decision Trees | 99.85 % | 97.57 % | 0.97 | 0.97 | 0.98 | 0.98 | 0.9757 |
| Random Forests | 99.83 % | 98.63 % | 0.99 | 0.97 | 0.98 | 0.99 | 0.9863 |
| HistGBM | 99.61 % | 98.70 % | 0.99 | 0.98 | 0.99 | 0.99 | 0.9870 |
| XGBoost | 99.86 % | 98.63 % | 0.99 | 0.98 | 0.99 | 0.99 | 0.9863 |
| Neural Networks (MultiLayer Perceptron) + logistic activation | 99.30 % | 98.37 % | 0.98 | 0.98 | 0.99 | 0.99 | 0.9837 |
| **Neural Networks (MultiLayer Perceptron) + ReLU activation** | 99.90 % | **99.03 %** | 0.99 | 0.98 | 0.99 | 1.00 | **0.9903** |

Table: - For model performance evaluation, the first column has the Training accuracy, while all the remaining columns are evaluated on Test data. The accuracy columns are reported in percentage, while all the other columns are reported in decimal.

The results demonstrate that Neural Networks have the best performance (99.03% accuracy) followed by SVM with Radial Basis Function Kernel (98.86%), and Ensemble methods like Random Forests (98.86%). It is also important and useful to validate that these models perform better than the baselines [14] established by KNN and logistic regression for android malware classification. The above results show the comparative performance of the models are similar to that observed in [4], [8], [11], [12] which also experiment on additional datasets. As expected, RBF SVM performs better than SVM with a linear kernel, and Ensemble models like Random Forests, HistGBM, XGBoost perform better than Decision Trees.

We also observe from the above results that neural networks with ReLU activation function have the best performance on the test data, so it is important to identify app features / key indicators used by a machine learning or deep learning model to classify an android application as 'malware'.

### 4.5 Explainability of Models and Feature Importance

An important aspect of machine learning and deep learning models is explainability [7] when deploying them for real-time android malware detection. Although certain machine learning models such as decision trees, random forests, etc. are easily interpretable through feature importances (Figures shown below), Neural Networks and Deep Learning methods are not easily interpretable. [7] uses Shapley Additive Explanation (SHAP) values to explain the classification model for android malware detection. So, we employ the SHAP summary plot to explain the key features used by the Neural Networks (MultiLayer Perceptron + ReLU activation) for successfully detecting android malware applications.

### 4.6 SHAP summary plot

We plot the SHAP summary plot only for the Neural Networks (MultiLayer Perceptron + ReLU activation) classifier as it has the best test performance compared to all other ML models.

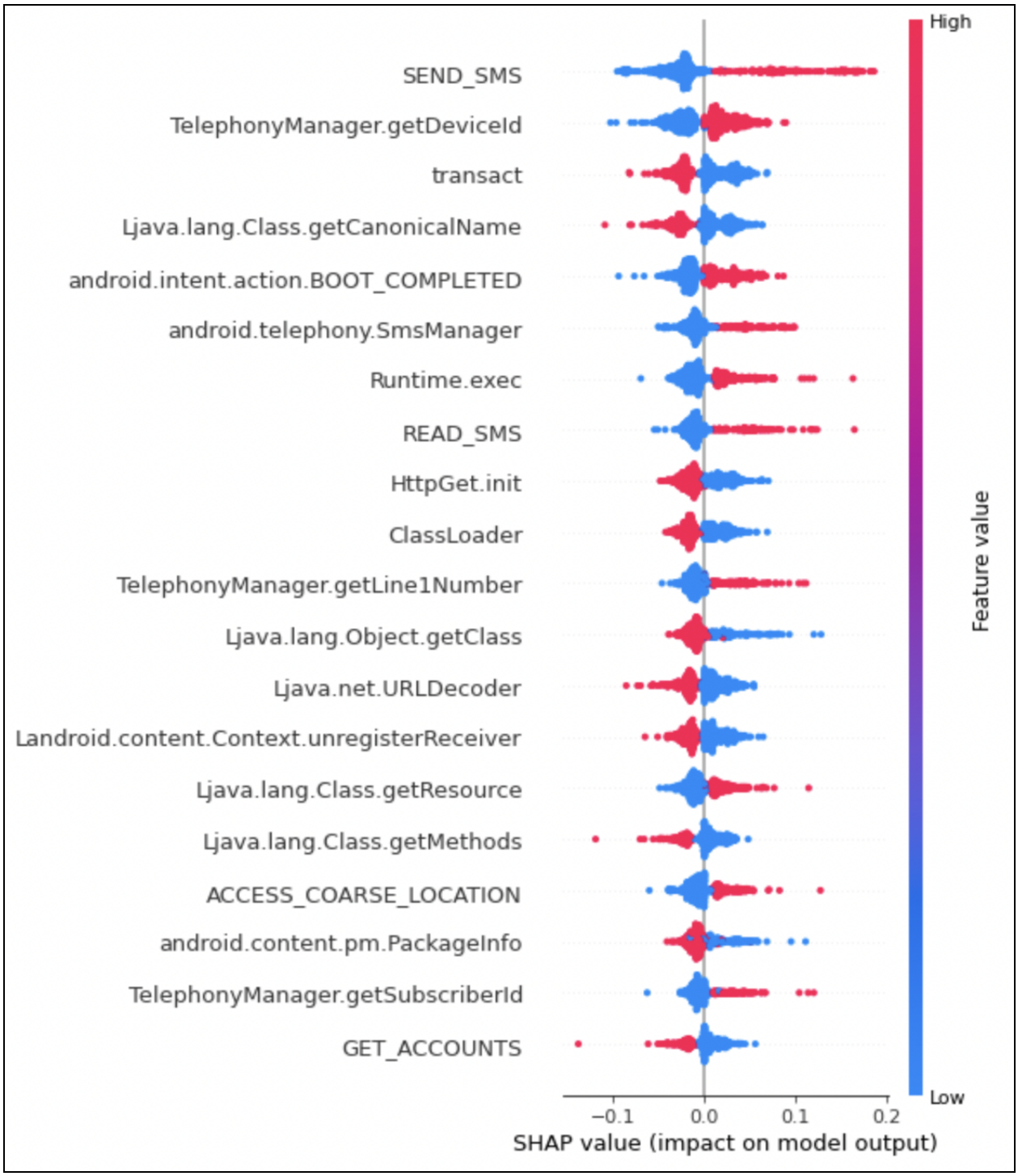
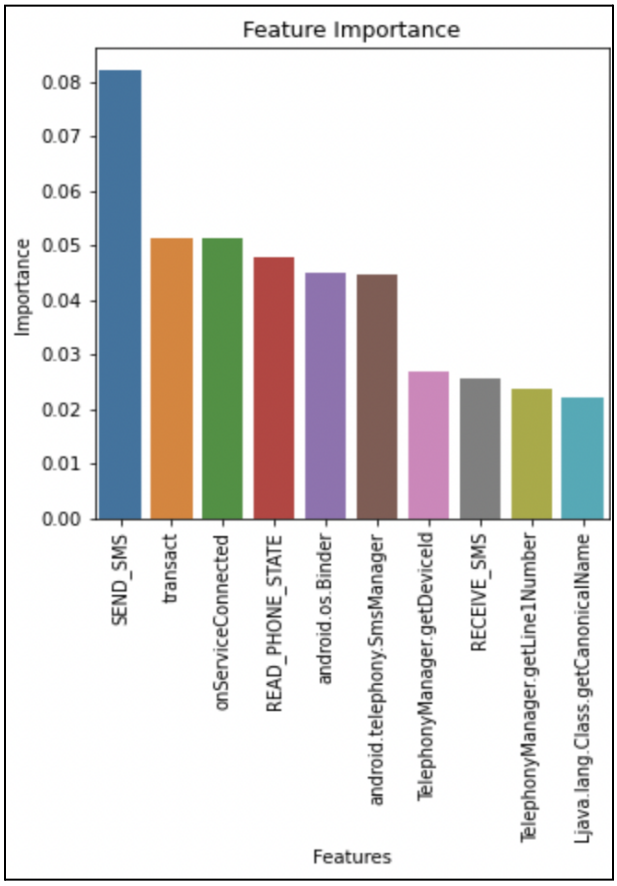



Figure:- As we can see from the SHAP plot - An app which utilizes the 'SEND_SMS', 'TelephonyManager.DeviceID' 'READ_SMS' have a strong correlation to being classified as 'Malware'; and typically apps which access 'Ljava classes', or 'transact' feature are more likely to be classified as benign. We also observe malware apps to access the device location, and have hidden 'runtime.exec' files as well. The Neural Network model uses these discriminative features with higher importance for detecting if an android application is benign or malicious.

### 4.7 Feature Importance plots

Random Forests




XGBoost

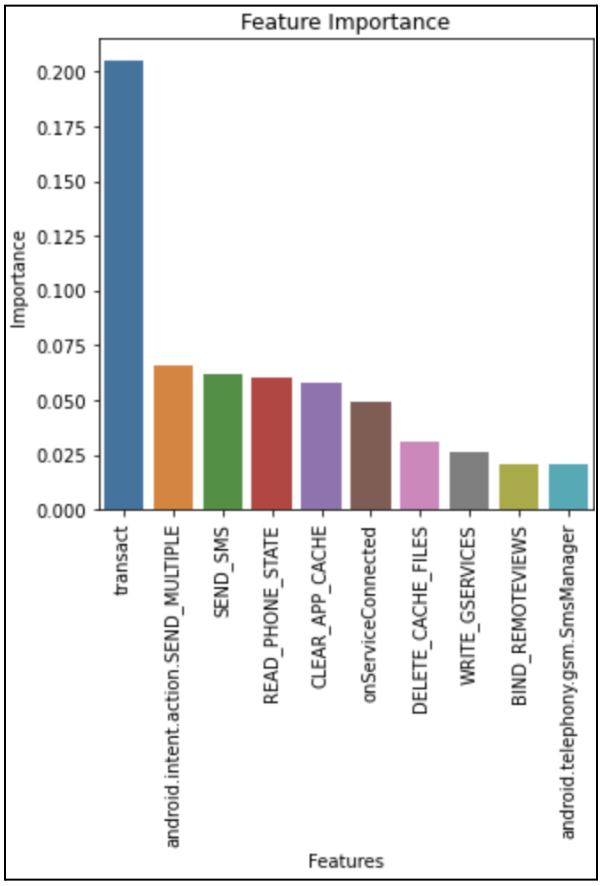



Figure: - Although these plots are are not as interpretable as the SHAP plots, we are able to infer that Random Forests and XGBoost models have a high importance to the following features - "SEND_SMS", "transact", "READ_PHONE_STATE". As we know from the previous SHAP plot that indeed "SEND_SMS", "transact" are both important features which help distinguish between malware and benign apps.

## 5. Conclusion

In this paper, we employed several machine learning models for android application malware detection and compared performance. The experimental results demonstrate that Neural Networks have the best performance (99.03% accuracy) followed by SVM with Radial Basis Function Kernel (98.86%), and Ensemble methods like Random Forests (98.86%). We also learned through SHAP explainability and feature importance plots that android applications that send or read sms, access device location and use hidden '.exe' files during run time are classified as Malware with a high-level of confidence and performance measure by our final Neural Network classifier.

### 5.1 What I learned from this literature study and experiment

We have been able to achieve 99.03% accuracy using Neural Networks achieving better performance than [8] on the same dataset (DREBIN-215). I learned the ML models can be successfully used for android app malware detection. Also I realized the importance of selecting features in a dataset and evaluating different machine learning models like KNN, Decision Trees, SVM, Ensemble methods etc. I learned the importance of tuning different hyper-parameters for different machine learning models. Although neural-networks are typically a black box, I learned the importance of using SHAP to better explain ML models and identified key discriminative features used by different models. In addition to learning the above technical skills, I learned how to conduct research in Machine Learning and Computer Security.

### 5.2 Limitations and Future Work

Although our experiment demonstrates that our Neural Network model achieves great performance over the certain state-of-the-art models, there is scope for further improvement. We can extend the scope of our model beyond binary classification to generalize for multi-class classification - detecting different malware types and families. Further research and work is needed to benchmark performance of the above machine learning and deep learning models for android malware classification on additional datasets like the AndrooZoo project which have more than 1 million samples.

## 6. References


[1] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun and H. Liu, "A Review of Android Malware Detection Approaches Based on Machine Learning," in IEEE Access, vol. 8, pp. 124579-124607, 2020, doi: 10.1109/ACCESS.2020.3006143. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9130686

[2] Y. Liu, C. Tantithamthavorn, L. Li and Y. Liu, "Explainable AI for Android Malware Detection: Towards Understanding Why the Models Perform So Well?," 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), Charlotte, NC, USA, 2022, pp. 169-180, doi: 10.1109/ISSRE55969.2022.00026. https://ieeexplore.ieee.org/abstract/document/9978981

[3] Sagar, R.; Jhaveri, R.; Borrego, C. Applications in Security and Evasions in Machine Learning: A Survey. *Electronics* 2020, *9*, 97. https://doi.org/10.3390/electronics9010097 https://www.mdpi.com/2079-9292/9/1/97

[4] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. 2020. A Survey of Android Malware Detection with Deep Neural Models. ACM Comput. Surv. 53, 6, Article 126 (November 2021), 36 pages. https://doi.org/10.1145/3417978 https://dl.acm.org/doi/abs/10.1145/3417978

[5] Google Developers, Android Manifest Overview, https://developer.android.com/ , Feb 22, 2023, https://developer.android.com/guide/topics/manifest/manifest-intro

[6] U. S. Jannat, S. M. Hasnayeen, M. K. Bashar Shuhan and M. S. Ferdous, "Analysis and Detection of Malware in Android Applications Using Machine Learning," 2019 International Conference on Electrical, Computer and

Communication Engineering (ECCE), Cox'sBazar, Bangladesh, 2019, pp. 1-7, doi: 10.1109/ECACE.2019.8679493. https://ieeexplore.ieee.org/document/8679493

[7] M. M. Alani and A. I. Awad, "PAIRED: An Explainable Lightweight Android Malware Detection System," in IEEE Access, vol. 10, pp. 73214-73228, 2022, doi: 10.1109/ACCESS.2022.3189645. https://ieeexplore.ieee.org/abstract/document/9825652

[8] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," in IEEE Transactions on Cybernetics, vol. 49, no. 2, pp. 453-466, Feb. 2019, doi: 10.1109/TCYB.2017.2777960. https://ieeexplore.ieee.org/document/8245867

[9] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch and C. Lu, "Towards Multimodal Learning for Android Malware Detection," 2019 International Conference on Computing, Networking and Communications (ICNC), Honolulu, HI, USA, 2019, pp. 432-436, doi: 10.1109/ICCNC.2019.8685502. https://ieeexplore.ieee.org/document/8685502

[10] M. E. Zadeh Nojoo Kambar, A. Esmaeilzadeh, Y. Kim and K. Taghva, "A Survey on Mobile Malware Detection Methods using Machine Learning," 2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2022, pp. 0215-0221, doi: 10.1109/CCWC54503.2022.9720753. https://ieeexplore.ieee.org/abstract/document/9720753

[11] E. J. Alqahtani, R. Zagrouba and A. Almuhaideb, "A Survey on Android Malware Detection Techniques Using Machine Learning Algorithms," 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 2019, pp. 110-117, doi: 10.1109/SDS.2019.8768729. https://ieeexplore.ieee.org/document/8768729

[12] J. Lopes, C. Serrão, L. Nunes, A. Almeida and J. Oliveira, "Overview of machine learning methods for Android malware identification," 2019 7th International Symposium on Digital Forensics and Security (ISDFS), Barcelos, Portugal, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757523. https://ieeexplore.ieee.org/document/8757523

[13] Chris Kuo, Explain Your Model with the SHAP Values, www.medium.com, September 13, 2019 https://medium.com/dataman-in-ai/explain-your-model-with-the-shap-values-bc36aac4de3d

[14] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Christian Wressnegger & Konrad Rieck 2021, Dos and Don'ts of Machine Learning in Computer Security. in Dos and Don'ts of Machine Learning in Computer Security. 2022 edn, 31st USENIX Security Symposium (USENIX Security 22), USENIX, Boston, MA, USA 2022. https://www.usenix.org/system/files/sec22-arp.pdf

# Appendix

**Table 10.** Malware Detection Analysis (Notations: LR-Logistic Regression, SMO-Sequential Minimal Optimal, MLP- Multi-Layer Perceptron).

| Reference | [135] | [136] | [137] | [138] | [139] |
|---|---|---|---|---|---|
| **Type of ML Algorithm** | J48, naïve bayes, SVM, LR, SMO, MLP | Linear SVM | Cascade one-sided perceptron, Cascade kernelized one-sided perceptron | Random Forest, Logistic Regression, SVM | SVM |
| **Advantage** | Able to get the high-level semantics of the malware | Able to protect user to install the application from an untrusted source | Non-stochastic version of algorithm enables parallelized training process to increase the speed | High detection accuracy against the kernel level rootkits and user level memory corruption report | Able to identify unknown families and behavior which are not present in the learning corpus |
| **Limitation** | Unable to detect kernel rootkits | When new code is loaded dynamic triggering is not possible | Accuracy is less when scaling up with the large datasets | For entire programme, Epochhistogram size should be chosen carefully which requires human effort | Relies on single program execution of malware binary |
| **Performance Metrics** | False positive rate, false negative rate, Accuracy- 99.7% | False detection, missing detection, accuracy- 93% | Sensitivity measure value, specificity measure value, accuracy measure value- 88.84%, True positives, false positives | False positives, true positives | Accuracy- 88%, confusion matrix |
| **Type(s) of Malware detected** | Backdoors, exploits, user-level rootkits, exploit, flooder, hack tools, net-Worm, Trojan, virus | Fake installer, DroidKungfu, Palnkton, opfake, GingerMaster, BaseBridge, Iconosys, Knim, FakeDoc, Geinimi, Adrd, DroidDream, LinuxLottor, GoldDream, MobileTx, FakeRun, Sendpay, Gappusin, Imlog, SMSreg | Backdoor, hack Tool, rootkit, Trojan, worms | Root kits | Worm, backdoors, trojans |
| **Type of features employed to the classifiers for detection** | Memory, network, file system, process- related system calls | Suspicious API calls, requests permissions, application components, filtered intents, network addresses, hardware features, used permission, restricted API calls | Binary type feature set | Architectural events, memory address, instruction mix | Frequency of contained string, string features (name & list of key-value pairs) |

Reference - [1], Liu et. al., A Review of Android Malware Detection Approaches Based on Machine Learning

The concepts of $FP$, $FN$, $TP$, and $TN$ are defined as follows.

(1) True positive ($TP$): the application is a malicious application and was correctly predicted to be malicious;

(2) False positive ($FP$): the application is not a malicious application but was wrongly predicted to be malicious;

(3) True negative ($TN$): the application is not a malicious application and was correctly predicted to be non-malicious;

(4) False negative ($FN$): the application is a malicious application but was wrongly predicted to be non-malicious.

The above four results are mutually exclusive, and thus their sum is the total number of samples in the test. Based on these four basic concepts, a series of performance metrics has been derived. Some commonly used metrics are as follows.

(1) Accuracy ($A_{cc}$) represents the ratio of correct predictions among the total number of samples in the test. Equation 1 shows how accuracy is computed.

$$A_{cc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

(2) Error Rate ($E_{rr}$) represents the ratio of false predictions among the total number of samples in the test. Equation 2 shows how error rate is computed.

$$E_{rr} = \frac{FP + FN}{TP + TN + FP + FN} \tag{2}$$

(3) Precision ($P$) represents the ratio of all samples correctly predicted to be positive among all samples predicted to be positive. Equation 3 shows how precision is computed.

$$P = \frac{TP}{TP + FP} \tag{3}$$

(4) Recall ($R$) represents the ratio of all positive samples correctly predicted among all positive samples. Equation 4 shows how recall is computed.

$$R = \frac{TP}{TP + FN} \tag{4}$$

Precision and recall are very important performance metrics, but they only provide a partial evaluation. In order to combine these two values to obtain a more complete evaluation of the performance of the classifier, the harmonic mean of precision and recall can be used, which is known as the $F_1$ score. Equation 5 shows how the $F_1$ score is computed.

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{5}$$

As mentioned in the experiments section, we use the Weighted F1 score to account for class imbalance in the dataset. [ Additional Reference - sklearn_f1_score )

**TABLE 2.** Summary of data preprocessing in selected references related to Android malware detection based on machine learning.

| Year | Reference | Description | Type of Data Preprocessing |
|---|---|---|---|
| 2012 | [99] | Control flow graphs (CFGs) with no more than 5 nodes are discarded because such CFGs are relatively rare and contain less information, and thus discarding them can greatly improve processing speed. | Data Cleaning |
| 2014 | [100] | Natural language processing (NLP) is used to filter and extract topic descriptions of the application. Applications with less than 10 topic words and without calling any sensitive APIs are removed, leaving 22,521 applications for subsequent clustering and analysis. | |
| 2017 | [101] | System calls that are never invoked by any application in the dataset are removed. | |
| 2018 | [93] | There are a lot of ambiguous vectors in the eigenvector matrix, which will affect the performance of the classifier. By removing these ambiguous vectors, the effect of classification can be improved. | |
| 2019 | [102] | Preprocess the application descriptions in the app store and then subject them to topic analysis. There are three phases to this process: (1) use regular expressions to remove non-text descriptions; (2) tokenize the description into a list of words, then remove punctuation and stop words; (3) reduce words to their root. | |
| 2011 | [103] | Basic information of the device, the list of installed applications, system calls, and other information are integrated into the feature vectors for subsequent analysis. | Data Integration |
| 2014 | [104] | The feature data that measures the complexity of a single method or class is aggregated through six functions to form a complete feature set of application complexity. | |
| 2019 | [105] | API information is extracted from a control flow graph (CFG); three types of data on the API are established and integrated into the feature set. The three types of data are API usage data, API frequency data, and API sequence data. | |
| 2015 | [106] | To reduce the dimensionality of feature vectors and avoid overfitting, features are kept as generic as possible by replacing specific identifiers of the application with tokens. | Data Reduction |
| 2017 | [107] | Principal component analysis (PCA) is used to reduce the feature space to a new feature space composed of a linear combination of components from the original features. | |
| 2018 | [108] | The affinity propagation (AP) clustering algorithm is used to replace the original data with a clustered representation of the data to reduce feature size. | |
| 2016 | [109] | The binary executables are disassembled into opcode sequences, and then converted into images. Histogram normalization and other methods are used to enhance the contrast between malicious and benign application images. | Data Transformation |
| 2017 | [110] | Researchers extract the APK file, convert the 4 files of classes.dex, AndroidManifest.xml, resources.arsc, and CERT.RSA4 into 8-bit unsigned integer vectors, organize them into a two-dimensional array, and finally visualize them as grayscale images. The grayscale images are decomposed by wavelet transformation, and image textures can be obtained for subsequent machine learning. | |
| 2018 | [111] | The binary files are used to generate grayscale images, and then a bilinear interpolation algorithm is used to preprocess the grayscale images so that the images have the same length and width, thereby making them suitable for the subsequent image classification by a convolutional neural network (CNN). | |
| 2014 | [112] | The API call sequences at runtime are extracted, represented in the form of n-grams, and finally normalized. | |
| 2015 | [113] | Dalvik instructions are represented in the form of n-grams, and the frequency of different n-grams is calculated for further processing. | |
| 2017 | [114] | Dalvik bytecode's n-gram form and the corresponding occurrence frequency are obtained as the feature vector. | |
| 2017 | [115] | Dalvik instructions are expressed in the form of n-grams. | |
| 2019 | [116] | The API call sequences are represented in the form of n-grams. | |
| 2010 | [117] | Raw data such as continuously measured data and events within Android applications are obtained by monitoring. Knowledge-based temporal abstraction (KBTA) is used to transform raw data into time-based features. | |
| 2016 | [118] | One type of feature is the co-occurrence matrix vector. The co-occurrence matrix is established based on the system call sequence and is then normalized and finally transformed into a vector. | |
| 2018 | [119] | Researchers convert the opcode sequence into a matrix vector, and transform the one-dimensional vector into a two-dimensional matrix, which is suitable for subsequent learning in a deep neural network (DNN). | |
| 2018 | [120] | The function call graphs (FCGs) extracted from an APK file are used to generate the topological signatures of the corresponding applications. | |

Reference - [1], Liu et. al., A Review of Android Malware Detection Approaches Based on Machine Learning