Android Applications Malware Analysis Using Machine Learning Based Approach

Kiran Shehzadi[1*], Hui Li[1], Shayan Zahid[2], Rafay Shabbir[3]

[1]Department of Computer Science and Technology, College of Information Science and Technology, Beijing University of Chemical Technology, China

[2]Department of Zoology, College of Entomology, Bahauddin Zakariya University, Pakistan

[3]Department of Computer Science, College of Cyber Security, Air University Islamabad, Pakistan

*Corresponding Author: kiranshahzadikh@gmail.com

DOI:10.30212/JITI.202312_1(1).0002

ABSTRACT

In the rapidly evolving domain of mobile security, effective detection of Android application malware is a critical challenge. In the emerging security industry, one of the biggest concerns is detecting malware in Android applications. This study explores how machine learning techniques can be used to identify malware in Android apps using a dataset called Drebin malware. We conducted an evaluation of machine learning models, including Random Forest, Naive Bayes, Artificial Neural Network, Perceptron, Sequential Neural Networks (NN), K-Nearest Neighbors (K-NN) Logistic Regression, and Support Vector Machines (SVM) with various kernels, like Radial Basis Function (RBF), Polynomial (Poly), and Linear as well as Decision Trees. The performance of these models was assessed based on accuracy, precision, recall, and F1-score. Our findings revealed that Random Forest and Artificial Neural Network models significantly outperformed the others, achieving accuracy rates of 98.77% and 98.57%, respectively. These models also outperformed others in terms of precision, recall and F1-score. While the Naive Bayes model showed efficiency compared to others, SVM with RBF kernel and Logistic Regression also demonstrated performance. This research highlights the capabilities of advanced machine learning algorithms such as Random Forest and ANNs when it comes to detecting Android malware within the Drebin dataset. The findings provide insights for enhancing cybersecurity measures, against the challenges presented by Android malware. These insights provide a valuable contribution to the field of cybersecurity, underscoring the effectiveness of machine learning in combating the sophisticated and evolving threats in Android malware.

Keywords: Static analysis, Android malware detection, Machine Learning approach, classifier, malware detection

1. Introduction

Android reigns supreme as the most ubiquitous mobile operating system, capturing the lion's share of the market. Malware encompasses a variety of harmful or intrusive software, including viruses, worms, backdoors, spyware, Trojan horses, and rootkits. The invention of the computer virus "Brain" in 1986 and its rapid dissemination, which resulted in thousands of computer systems being

damaged. Since computerization was still in its early stages, malware development at that time was progressing slowly [1]. However, things have changed, and every day, a thousand new malware programs are created. Around 86.6% of smartphones sold worldwide in 2019 were Android-based. As of the end of April 2020, Google Play—the official Android app store—had over 2.8 million applications available [2]. The International Association of Mobile Operators (GSMA) published a study predicts that by the end of 2020, there will be more than 8 billion mobile phone users globally, up from 5,600 million at the end of 2016[3]. With over 65 billion app downloads from Google Play, Android has established itself as the most popular mobile operating system, with over 1 billion devices sold worldwide. Malware has also found a way to attack Android due to its increasing popularity and the proliferation of third-party apps shops [4]. Threat vectors related to Android devices have grown along with their exponential surge in popularity. Malicious Android apps have the potential to create a wide range of problems, such as data theft, user settings manipulation, service availability and integrity breach [5]. Avast's research observed that cyberattacks alongside android operating system are increasing by 40% year-over-year since 2016[6]. So, it is necessary to verify and enhance the malware detection methods that are currently in use [7].

Various approaches have been proposed to enhance the accuracy of malware detection systems. In recent years, machine learning has emerged as a powerful tool for Android malware detection. As a branch of artificial intelligence, machine learning enables systems to automatically learn and extract patterns from vast amounts of data. This capability makes machine learning well-suited for identifying malicious applications based on their characteristics and behavior [8]. The precision rates of the Machine Learning approaches diverge based on the quality of the extracted features. Despite the growing popularity of Machine Learning/ Deep Learning approaches as primary detection algorithms against malicious attacks, malware families continue to expand at an alarming rate[9]. While numerous studies and investigations in this area are still ongoing, the effectiveness of current detection methods remains a significant concern [10]. As mobile development progresses towards a more interconnected future, enhanced security measures are crucial to mitigate the escalating threat of malware [11]. To effectively safeguard mobile platforms, more comprehensive prevention strategies must be implemented.

This Paper aims to add to the existing malware detection efforts by highlighting the distinctions between methodologies and introducing a high-efficiency and high-accuracy detection solution. Using static analysis permissions, Application Programming Interface (API) was called to detect malware, based on many models of ML / DL algorithms, Random Forest, Sequential Naive Bayes, Neural Network, Logistic Regression, Artificial Neural Network, Perceptron, Decision Tree (DT) and Support Vector Machine (SVM linear, SVM polynomial, SVM Radial basis function-RBF), Kernal's and K-nearest neighbors (K-NN) [12]. Dataset used named Drebin-215-dataset-5560 malware-9476-benign.csv com. Utilizing the Derbin dataset enables researchers to assess the effectiveness of their detection algorithms against a wide range of malware families and variants. This paper contributes to the advancement of Android malware detection by introducing a novel static analysis-based method that utilizes the Drebin dataset [13]. The paper meticulously defines and categorizes the four classes of malware within the dataset, laying the groundwork for achieving high

detection rates. By leveraging Drebin dataset, the proposed method effectively identifies malicious applications, enhancing the overall security posture of the Android ecosystem.

2. Literature Review

The invention of the computer virus "Brain" in 1986 and its rapid propagation caused hundreds of computer systems to be damaged, thereby beginning the era of malware development. Since computerization was still in its early stages, malware development at that time was progressing slowly. However, times have changed, and every day, a thousand new pieces of malware are created. In this research, we investigate the efficacy of machine learning classifiers for Android malware detection as the proposed model surpasses the conventional techniques in terms of accuracy and detection rates, showcasing the efficacy of machine learning algorithms in discerning malicious traffic. Novel feature sets address the issues (identified in previous studies) in android applications malware detection and improved its accuracy [14]. Previous studies have proposed adversarial attacks and defense strategies for android malware detection, but they have achieved limited fooling rates, and have not discussed the adversarial robustness of the proposed defense. The proposed work in this paper addresses the development of adversarial robust Android malware detection models and scrutinizes their resilience against adversarial attacks [15]. This study investigates the application of machine learning algorithms, specifically the K-Nearest Neighbor (K-NN) algorithm, in the context of Android malware detection. The analysis emphasizes the utilization of two principal sources of information for discerning Android malware, namely static and dynamic analysis. The research evaluates the efficacy of the K-NN algorithm by assessing key performance metrics such as accuracy, recall, F1-score, and precision. The evaluation was conducted using CICMalDroid 2020 dataset, which encompasses a comprehensive collection of both benign and malware samples characterized by static and dynamic features [16].

Determine which spare permissions have been sought, then utilize this information in the security and privacy strategy, which applies code and static analysis to the current datasets, compares the results, and establishes the accuracy level. The accuracy rate for classification is 91.95% [17].

A comparative analysis of diverse Android malware detection systems, elucidating their respective detection methodologies, analytical procedures, and feature extraction approaches. The assessment is conducted utilizing two distinct datasets, namely Malgenome and Maldroid, and incorporates the application of four discrete machine learning models: K-nearest neighbor, Naive Bayes, Support Vector Machine, and Decision Tree classifiers. The proposed system integrates feature extraction, normalization, standardization, and anomaly detection mechanisms. Furthermore, it integrates dynamic analysis to monitor system API calls and permissions during the execution of applications. The increasing threat of Android malware necessitates an effective detection system in both the business and academic fields [18]. The significance of automated Android applications malware detection is obvious because of the increasing number of applications and the impossibility of human inspection for each application. This paper proposes utilizing tree-based machine learning algorithms, particularly Random Forest and Xgboost, for static malware detection based on the requested permissions of applications. The proposed existing models (in terms of accuracy) demand less computational power and operate more efficiently [19]. They employ machine learning

classifiers based on categories to improve the efficiency of classification models in identifying harmful applications that fall into a particular category. Extensive machine learning experiments have demonstrated that category-based classifiers report a remarkably greater average performance compared to non-category-based classifiers. Malware ML classifier performance and detection accuracy are directly impacted by the feature space [20].

Proved that, with weak assumptions on the data corruption models, a far more robust SVM learning model may be created. Under a broad variety of attack parameters, our optimal SVM learning technique yields more robust overall performance when obtained under the constrained assault model (95.9%) – a false positive rate of 2.4%, precision rate 97.3%, accuracy 96.8%, and precision 97.3%; J48 decision tree had the greatest overall performance among the five classifiers, according to a study of test and experimental findings [21]. ML-based algorithms demonstrated higher level of accuracy in android malware detection when compared to other existing methods. Therefore, machine learning techniques for malware detection must be made available for Android smartphones. Many researchers presented various machine learning systems to detect malware based on ML. The experimental results obtained using support vector machines (SVM) and k-nearest neighbors (K-NN) classifiers on a dataset of real malware and benign apps reveal an average accuracy rate of 79.08% for SVM and 80.50% for K-NN [22].

We assessed the malware detection efficacy of machine learning classifiers and provided new feature sets that address the issue of earlier research in mobile malware detection [23].

Additionally, the average true positive rate, which represents the proportion of genuine malware correctly identified, was 67.00% for SVM and 80.00% for K-NN. These findings indicate that both SVM and K-NN are effective classifiers for malware detection, with K-NN exhibiting slightly superior performance [24]. Machine learning algorithms, including Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbor (K-NN), and Naive Bayes (NB), have been extensively employed in malware detection. The paper also explores feature extraction techniques, encompassing static extraction from executable APK files and dynamic extraction through monitoring application behavior in a simulated environment. A comparative analysis with existing literature reveals that the proposed GA-Stacking method surpasses the detection performance of other classifiers and integration models [25].

3. Research Design

3.1 Material and Methodology

This architectural diagram of machine learning model (Figure.1) illustrates the initial stage, commencing with the preprocessing of data derived from both malicious and benign APK files taken from a Derbin dataset which is available publicly for everyone to use. Then, applying the Python script developed in VS-Code to extract static features, including permissions and API calls, the extracted features are formatted and stored as a Comma Separated Values (CSV) file for training purposes. Then, further continue the process: we test the machine learning models' classification after determining the relevant metrics for the evaluation.

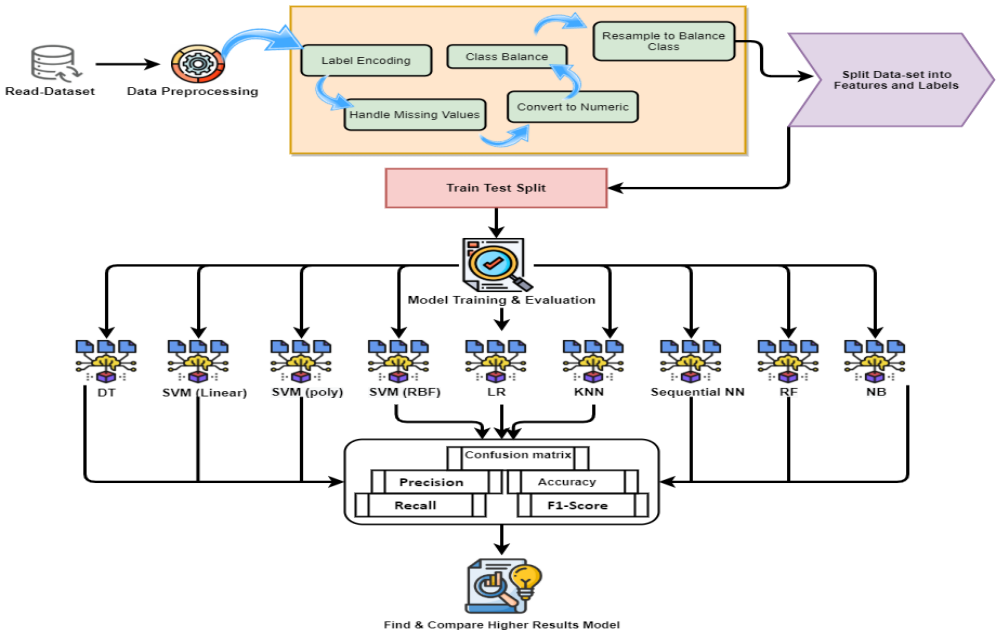


Figure1. Architecture of the Machine learning model
Source: By authors.

3.2 Dataset

The dataset employed in this study, known as the Drebin dataset and denoted as drebin.csv, encompasses features derived from 5,560 malicious and 9,476 benign Android applications, collectively representing 179 distinct malware families. These families include Adware, Ransomware, Scareware, Trojans, Backdoors, Botnets, and SMSware, each characterized by unique attack behaviors corresponding to their nomenclature. Notably, Adware and SMSware manifest intrusive advertisements and SMS messages during application execution, whereas Scareware and Ransomware demand payments or ransoms to avert potential system damage or data loss. Each data point within the Drebin dataset denotes an Android application characterized by a set of features, accompanied by a class label designating its malicious or benign classification. Originating in 2012, the Drebin dataset stands as a widely utilized repository of Android malware applications, specifically curated for research purposes in the domain of malware detection. The dataset encompasses samples gathered between August 2010 and October 2012, generously provided to the research community through the Mobile Sandbox project.

3.3 Data Preprocessing

Preceding the training of the model, the dataset underwent a series of preprocessing steps to optimize its compatibility with machine learning algorithms. The class labels were encoded numerically using label encoding. Entries containing special characters or missing values were identified and removed to maintain data integrity. Additionally, all features were converted to

numerical data types, enhancing their compatibility with machine learning algorithms and facilitating streamlined processing.

3.4 Feature Analysis

A correlation heatmap was generated to identify and visualize correlations between the various features. This analysis helped in understanding the relationships within the data and informed the feature selection process.

3.5 Class Balancing

The initial analysis revealed a class imbalance within the dataset. To address this, the minority class was oversampled to match the number of samples in the majority class, creating a balanced dataset that prevents bias towards the more represented class.

3.6 Model Development and Evaluation

Seven machine learning models were developed and evaluated; Decision Tree Classifier, Logistic Regression (LR), Random Forest (RF), Sequential Neural Network, Naive Bayes (NB), K-Nearest Neighbors (K-NN), Support Vector Machine (SVM). Three different SVM models were trained with: SVM linear, SVM polynomial, and SVM radial basis function (RBF) kernels to investigate the impact of kernel choice on model performance. All models were trained on an 80-20 train-test split of the balanced dataset. The training process incorporated a random state for reproducibility of results.

3.7 Model Evaluation Metrics

Model performance was evaluated using accuracy and confusion matrices. Accuracy provided a straightforward measure of model performance, while confusion matrices offered detailed insights into the true positive (TP) and negative rates, as well as false positive (FP) and negative rates. Various metrics computed assess the classifiers and determine the optimal classifiers that can yield favorable results. The performance assessment which involves a comprehensive evaluation of accuracy, F-measure, Recall, and Precision scores, was determined using the equations outlined is below.

$$Precision = \frac{TruePositive}{(TurePositve+FalsePositive)} \qquad Recall = \frac{TruePositive}{(TruePositive+FalseNegative)}$$

$$Accuracy = \frac{TP+TN}{(TP+TN+FP+FN)} \qquad F-Measure = \frac{2\times(Precision\times Recall)}{(Precision+Recall)}$$

Accuracy serves as a metrics for assessing classification models, representing proportion to correct predictions made by the model. Precision is defined as the ratio of accurately identified positive outcomes, total number of positive results, inclusive of those not correctly identified. Additionally, Recall is characterized by the correctly ratio of identified outcomes to the total number of samples that should have been accurately recognized. Precision can be interpreted as a gauge of classifier's exactness, with low Precision rate indicating a higher occurrence of false positives (FP). Conversely, recall gauges the completeness of classifiers, with low Recall suggesting a higher incidence of false negatives. F-Measure provides a measure of balance between Recall and Precision that can be utilized to select a model in order to achieve a balance between these two metrics.

3.8 Mathematical Equations

Here are the common mathematical operations used in machine learning models and in the code: Import and Seed Setting Cell; Importing libraries: pandas (pd), *numpy* (np), TensorFlow $(tf), keras,$and matplotlib; Setting random seeds for reproducibility: $np.random.seed(0)$ *and* $tf.compat.v1.set\_random\_seed(0)$. The equation.1 shows that: this cell performs data balancing by oversampling the minority class in the dataset. The process involves: Separating features (X) and labels (**y**), Identifying majority and minority classes based on value counts, Resampling the minority class to match the number of samples in the majority class, Combining the resampled minority class with the majority class data, Shuffling the balanced dataset for randomness. The mathematical representation for resampling can be expressed as:

$$Balanced\ Dataset\ Oversample\left(Minority\ Class, N_{majority}\right)\ldots\ldots \quad [1]$$

In Equation (1)where $N_{majority}$ is the number of samples in the majority class.: this program uses $train\_test\_split$ from $sklearn.model\_selection$ to divide the dataset into training and testing sets. The mathematical representation for this operation is:

$$Train, Test = Split(D, test_{size} = 0.2\ldots\ldots\ldots\ldots\ldots\ldots \quad [2]$$

The equation number (2) where D is the dataset and $Test\_size = 0.2, Test\_size = 0.2$ indicates that 20% of the data is reserved for testing. The program includes implementations of various machine learning models. Here's a summary of the key models found and their mathematical representations. Common formulas used for different models like Neural Network can be represented as a series of functions:

$$NN(x) = f_n(\ldots f_2(f_1(x, W_1, b_1), W_2, b_2) \ldots, W_n, b_n)\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad [3]$$

where f_i represents "activation function" of the $i-th$ layer, W_i and b_i are the "weights" and "biases of the layer" and x is the input. SVMs, possibly with different kernels like Radial Basis Function (RBF) linear and Polynomial (Poly), are mentioned and the basic formulation of an SVM (for linear kernel) is:

$$minimize\ \frac{1}{2}\ ||w||^2\ subject\ to\ y_i(w.x_i + b) \geq 1, for\ all\ i\ \ldots\ldots\ldots\ldots\ldots \quad [4]$$

where w is the "weight" vector, b is the "bias", x_i are the training samples, and v_i are the label. Logistic Regression seems to be used as a classifier. Its mathematical form is:

$$P(x) = \frac{1}{1+e^{-(w.x+b)}}\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad [5]$$

where $P(y=1|x)$ is the probability that the class label y is 1 given the features x, w is the weight vector, and b is the bias.

3.9 Graphical Explanation

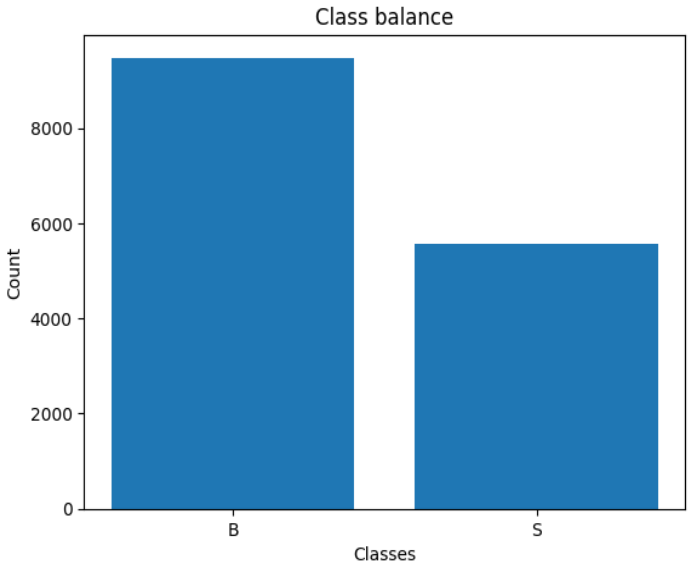


Figure 2. Bar chart
Source: By authors.

The presented visualization takes the form of a bar chart entitled "Class Balance," delineating the distribution of counts between two classes, denoted as "B" and "S." Examination of the chart reveals that the count for class "B" exceeds 8000, while class "S" registers a count slightly below to 6000. Bar charts of this nature are commonly employed to compare the magnitudes of distinct groups. In this specific instance, the graphical representation underscores an imbalance between classes, with class "B" demonstrating a higher count than class "S." This observation holds significance in diverse contexts, particularly within the realm of data analysis for machine learning, where achieving and maintaining class balance is imperative for optimizing model performance. Notably, a pronounced numerical disparity between classes may introduce bias into the model, favoring the more prevalent class.

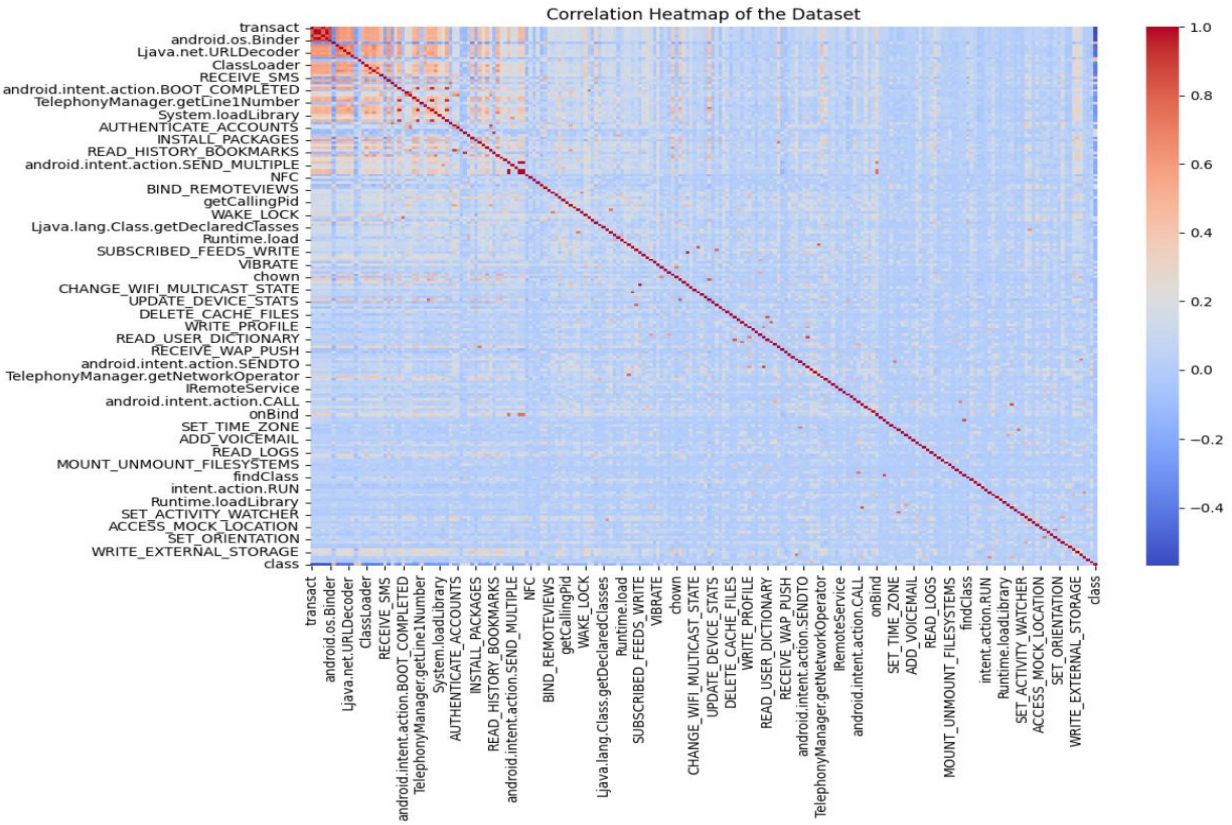


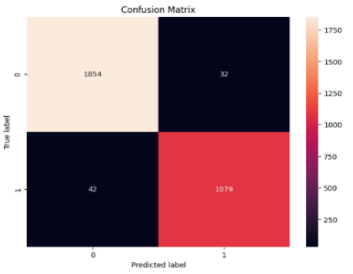
Figure 3. Correlation Heatmap
Source: By authors.

The matrix includes various permissions and features (like $android.os.Binder$, READ_SMS, ACCESS_WIFI_STATE, etc.) that are likely associated with an Android application dataset. Each square (or pixel) in the heatmap corresponds to the correlation between the permissions/features on the X-axis and the Y-axis. The color scale on the right indicates the strength and direction of the correlation:

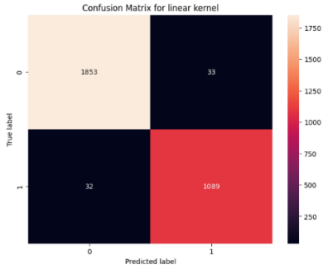
- A correlation of 1 (dark red) means a perfect positive correlation, meaning when one permission is used, the other is also used.
- A correlation of 0 (blue) indicates no correlation; the permissions/features do not tend to be found together.
- A negative correlation (light blue to dark blue) indicates an inverse relationship; as one permission is used, the other is less likely to be used.

The diagonal line from the top left to the bottom right shows the correlation of each permission with itself, which is always 1 (perfect correlation).

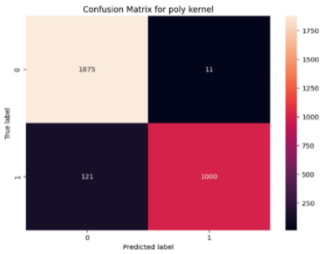
3.10 Confusion Matrix



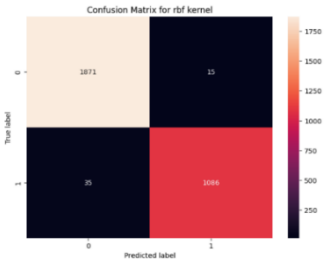
Decision Tree (a)



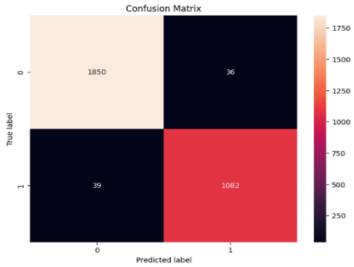
SVM Linear (b)



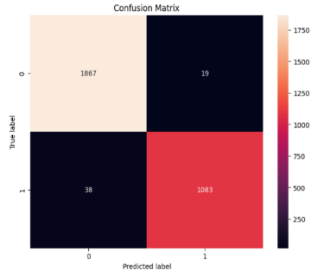
SVM Poly Kernel (c)



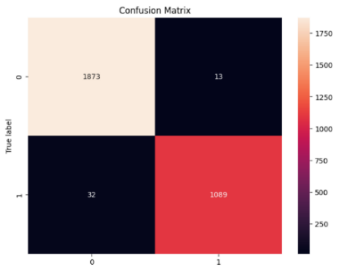
SVM RBF (d)



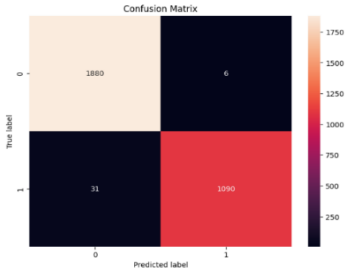
Logistic Regression (e)



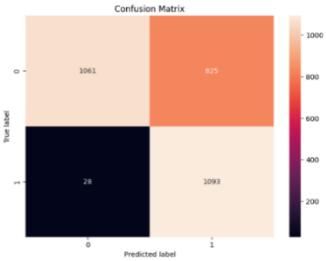
K-Nearest Neighbors (f)



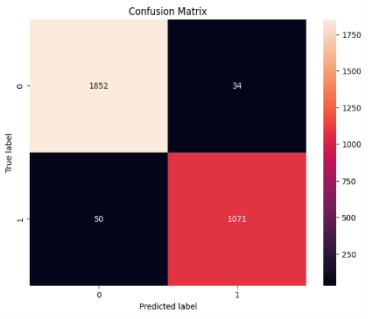
Sequential NN 4(g)



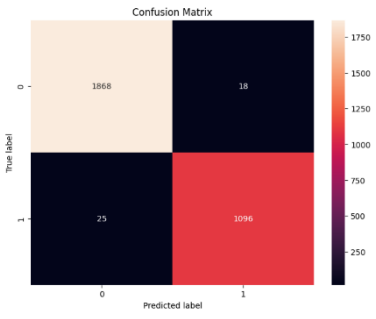
Random Forest (h)



Naive Bayes (i)



Perceptron (j)



ANN (k)

Figure 4. represents Confusion Matrix, which is a performance measurement for machine learning classification applied in research. It compares the actual target values with those predicted by the model in this matrix.

Source: By authors.

3.10.1 Decision Tree (DT)

In Figure. 4 (a), Top-Left Square (True Negative - TN): The model predicted '0', and the true label was '0'. There were 1854 cases where the model is correctly predicted to the negative class. Top-Right Square (False Positive): The model predicted '1', but the true label was '0'. There were 32 cases where the model incorrectly predicted the positive class. Bottom-Left Square (False Negative - FN): The model predicted '0', but the true label was '1'. There were 42 cases where the model incorrectly predicted the negative class. Bottom-Right Square (True Positive - TP): The model predicted '1', and the true label was '1'. There were 1079 cases where the model correctly predicted the positive class.

3.10.2 SVM Linear Kernel

In figure. 4 (b), Top-Left Square (True Negative - TN): The number here, 1853, indicates the instances where the model correctly predicted the negative class (label '0'). Top-Right Square (False Positive - FP): The number 33 represents the instances where the model incorrectly predicted the positive class (label '1') when the true label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 32 represents the instances where the model incorrectly predicted the negative class (label '0') when the true label was positive (label '1'). Bottom-Right Square (True Positive - TP): The number 1089 represents the instances where the model correctly predicted the positive class (label '1').

3.10.3 SVM Poly Kernel

In figure.4 (c), Top-Left Square (True Negative - TN): The number 1875 represents the cases where the model correctly predicted the actual negative class (label '0'). Top-Right Square (False Positive - FP): The number 11 represents the cases where the model incorrectly predicted the positive class (label '1') for actual negatives (label '0'). Bottom-Left Square (False Negative - FN): The number 121 represents the cases where the model incorrectly predicted the negative class (label '0') for actual positives (label '1'). Bottom-Right Square (True Positive - TP): The number 1000 represents the cases where the model correctly predicted the actual positive class (label '1').

3.10.4 SVM RBF Kernel

In figure. 4 (d), Top-Left Square (True Negative - TN): The number 1871 indicates the instances where the classifier correctly predicted the negative class (label '0'). Top-Right Square (False Positive - FP): The number 15 represents the instances where the classifier incorrectly predicted the positive class (label '1') when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 35 represents the instances where the classifier incorrectly predicted the negative class (label '0') when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The number 1086 represents the instances where the classifier correctly predicted the positive class (label '1').

3.10.5 Logistic Regression

In figure. 4 (e), Top-Left Square (True Negative - TN): The number 1850 indicates that the model correctly predicted the negative class (label '0') 1850 times. Top-Right Square (False Positive - FP): The number 36 indicates that the model incorrectly predicted the positive class (label '1') 36

times when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 39 indicates that the model incorrectly predicted the negative class (label '0') 39 times when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The number 1082 indicates that the model correctly predicted the positive class (label '1') 1082 times.

3.10.6 K-Nearest Neighbors

In figure. 4 (f), Top-Left Square (True Negative - TN): The number 1867 indicates that the model correctly predicted the negative class (label '0') 1867 times. Top-Right Square (False Positive - FP): The number 19 indicates that the model incorrectly predicted the positive class (label '1') 19 times when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 38 indicates that the model incorrectly predicted the negative class (label '0') 38 times when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The number 1083 indicates that the model correctly predicted the positive class (label '1') 1083 times.

3.10.7 Sequential Neural Networks

In figure. 4 (g), Top-Left Square (True Negative - TN): The number 1873 represents the true negatives, which means that the model correctly predicted the negative class (label '0') 1873 times. Top-Right Square (False Positive - FP): The number 13 is the count of false positives, where the model incorrectly predicted the positive class (label '1') when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 32 represents the false negatives, where the model incorrectly predicted the negative class (label '0') when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The number 1089 is the count of true positives, indicating that the model correctly predicted the positive class (label '1') 1089 times.

3.10.8 Random Forest

In figure. 4 (h), Top-Left Square (True Negative - TN): The number 1880 represents the true negatives, which are the instances where the model correctly predicted the negative class (label '0'). Top-Right Square (False Positive - FP): The small number 6 represents the false positives, where the model incorrectly predicted the positive class (label '1') when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 31 represents the false negatives, where the model incorrectly predicted the negative class (label '0') when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The large number 1090 represents the true positives, which are the instances where the model correctly predicted the positive class (label '1').

3.10.9 Naive Bayes

In figure. 4 (i), Top-Left Square (True Negative - TN): The number 1061 signifies the instances where the model correctly predicted the absence of the condition (class '0'). Top-Right Square (False Positive - FP): The number 825 indicates the instances where the model incorrectly predicted the presence of the condition (class '1') when it was actually absent (class '0'). Bottom-Left Square (False Negative - FN): The number 28 indicates the instances where the model incorrectly predicted the absence of the condition (class '0')) when it was actually present (class '1'). Bottom-Right Square

(True Positive - TP): The number 1093 signifies the instances where the model correctly predicted the presence of the condition (class '1').

3.10.10 Perceptron

In figure. 4 (j), Top-Left Square (True Negative - TN): The number 1852 signifies the instances where the model correctly predicted the absence of the condition (label '0'). Top-Right Square (False Positive - FP): The number 34 indicates the instances where the model incorrectly predicted the presence of the condition (label '1') when it was actually absent (label '0'). Bottom-Left Square (False Negative - FN): The number 50 indicates the instances where the model incorrectly predicted the absence of the condition (label '0') when it was actually present (label '1'). Bottom-Right Square (True Positive - TP): The number 1071 signifies the instances where the model correctly predicted the presence of the condition (label '1').

3.10.11 Artificial Neural Network

In figure. 4 (k), Top-Left Square (True Negative - TN): The number 1868 represents the true negatives, which are the instances where the model correctly predicted the negative class (label '0'). Top-Right Square (False Positive - FP): The small number 18 represents the false positives, where the model incorrectly predicted the positive class (label '1') when the actual label was negative (label '0'). Bottom-Left Square (False Negative - FN): The number 25 represents the false negatives, where the model incorrectly predicted the negative class (label '0') when the actual label was positive (label '1'). Bottom-Right Square (True Positive - TP): The large number 1096 represents the true positives, which are the instances where the model correctly predicted the positive class (label '1').

4. Results and Discussion

The evaluation of the models on the test set revealed the following Accuracy outlined in figure.5:

- Decision Tree Classifier: Demonstrated a robust performance with an accuracy of 97.54%.
- Support Vector Machine (SVM): The SVM with a linear kernel achieved an accuracy of 97.84% while the polynomial kernel achieved 95.61%, and the RBF kernel achieved 98.34% accuracy.
- Logistic Regression: Performed comparably well with an accuracy of 97.51%.
- K-Nearest Neighbors (K-NN): Recorded an accuracy of 98.10%, indicating high similarity among neighbors in the feature space.
- Sequential Neural Network: Surpassed the traditional models with the highest accuracy of 98.50%, signifying the efficacy of neural networks in capturing complex, non-linear patterns in the data.
- Artificial Neural Network: performance of its deep learning model achieved an accuracy of 98.57%.
- Perceptron: Demonstrated performance with an accuracy of 97.21%.
- Naive Bayes: Demonstrated performance with an accuracy of 71.63%.
- Random Forest: Demonstrated performance with an accuracy of 98.77%.

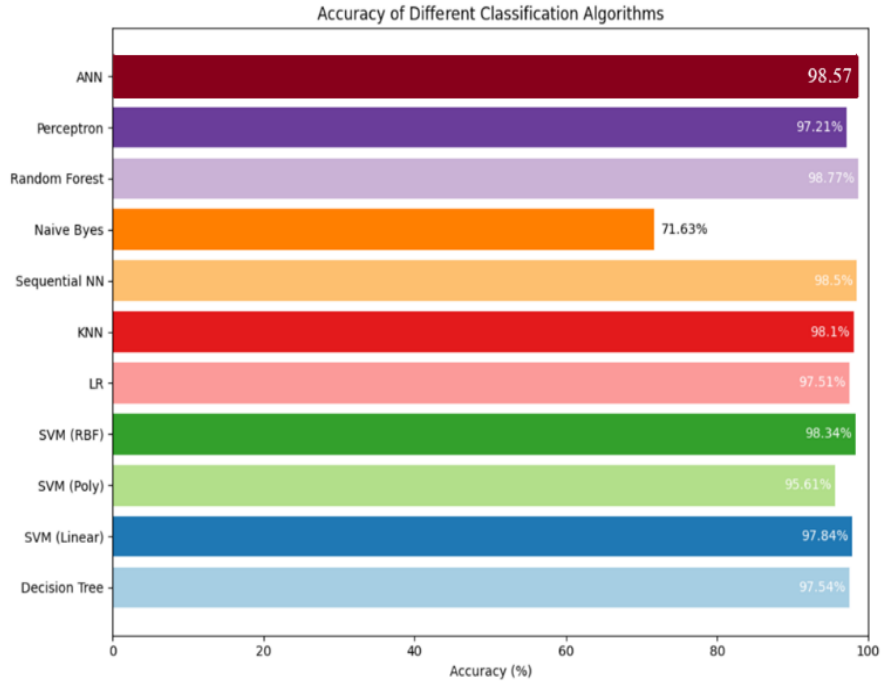


Figure 5. Accuracy of Different Algorithm Classification
Source: By authors.

Table 1. The performance of the tested algorithms in detecting Android malware

ML-Models	Accuracy	Precision	Recall	F1-Score
Random Forest	98.77%	98.92%	98.46%	98.68%
Naive Bayes	71.63%	77.21%	76.88%	71.63%
Sequential NN	98.50%	98.57%	98.23%	98.39%
K-NN	98.10%	98.14%	97.80%	97.97%
Logistic Regression	97.51%	97.36%	97.31%	97.33%
SVM(RBF)	98.34%	98.40%	98.04%	98.22%
SVM(Poly)	95.61%	96.42%	94.31%	95.20%
SVM(Linear)	97.84%	97.63%	97.70%	97.69%
Decision Tree	97.54%	97.45%	97.28%	97.36%
ANN	98.57	98.53	98.41	98.47
Perceptron	97.21	97.51	96.87	97.00

Source: By authors.

The combination of these elements contributes to a robust methodology that is tailored to the challenges of malware detection in Android applications. The results of this research may provide a benchmark for future studies and contribute to the development of more effective malware detection

systems. In this Research, we conducted experiments on two types of models; traditional machine learning (ML) classifiers and deep learning (DL) models; Namely: Random Forest, Naive Byes, Sequential NN, K-NN, SVM with (RBF), SVM with (Poly), SVM with (Linear), Decision Tree (DT), Artificial Neural Network (ANN), Perceptron and Logistic Regression (LR). The trained classifiers on the dataset proceed to perform testing and evaluation, in both experiments, utilizing features extracted from API calls and permissions. These features are straightforward to implement and apply, and they can be effectively employed, both regression tasks and classification. Table 1 presents the performance of the classifiers in Android malicious applications detection. The Random Forest (RF) classifier emerged as the top performer, achieving an impressive accuracy of 98.77%, compared to other models, higher rate of correct predictions at 98.92%, and an F1-score indicating its ability to correctly predict 98.68% of the dataset. Additionally, the RF classifier demonstrated remarkable recall, correctly predicting 98.46% of malware samples. Moreover, it exhibited the highest precision, identifying 98.92% of the entire dataset during the testing process. On the other Hand, Artificial Neural Network (ANN) models significantly outperformed the others, achieving accuracy rates of 98.57% and then Sequential Neural Network achieving an accuracy of 98.50%. The Precision Rate of Sequential Neural Network (98.57%) is higher compared to the ANN model (98.53%). However, the Recall rate (98.41%) and F1-score indicating its ability to correctly predict 98.47% of the dataset is higher compared to Sequential Neural Network (Recall 98.23% and the F1-score correctly predict 98.39%). While the Naive Bayes model showed low efficiency compared to others, SVM with RBF kernel and Logistic Regression also demonstrated an excellent performance. At the end, we achieved higher result from Random Forest, Artificial Neural Network, and Sequential Neural Network.

5. Conclusions

In this research, we investigated the android malware detection effectiveness of machine learning (ML) classifiers and suggested new feature sets that address the issue of previous studies in smart-mobile malware detection and improve its accuracy. We conducted an analysis of the top models used for classifiers (Random Forest, Naive Bayes, Sequential NN, ANN, Perceptron, K-NN, SVM with RBF, SVM with Polynomial, SVM with Linear, Logistic Regression, Decision Tree (DT). Random Forest achieved the highest accuracy over all (98.77%), the highest rate of correct predictions at 98.92%, and an F1-score indicating its ability to correctly predict 98.68% of the dataset. However, Artificial Neural Network models achieved an accuracy rate of 98.57% and then Sequential Neural Network achieving accuracy of 98.50%. However, ANN Recall rate (98.41%) and F1-score indicating its ability to correctly predict 98.47% of the dataset are higher compared to Sequential Neural Network (Recall 98.23% and F1-score correctly predict 98.39%). Additionally, the RF classifier demonstrated remarkable recall, correctly predicting 98.46% of malware samples. In future research direction, we will use ensemble models of machine learning approaches to get higher results with different models and different matrices.

Acknowledgements

Competing Interests: The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. Author contribution: Methodology and formal analysis are done by Kiran Shehzadi and Hui Li; data collection and English polishing are done by Shayn Zahid; final revision is done by Rafay Shabbir. All authors have read and agreed to the published version of the manuscript. Funding Information: No funding; Data Availability Statement: Not Available; Research Involving Human and /or Animals: No Human and /or Animals Involved. Informed Consent**:** Original research, we have not used another researcher's data.

References

[1] Yang, S., Wang, Y., Xu, H., Xu, F. and Chen, M. An android malware detection and classification approach based on contrastive lerning. Computers & Security, 2022, 123, 102915. DOI: 10.1016/j.cose.2022.102915.

[2] Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D. and Liu, H. A Review Of Android Malware Detection Approaches Based On Machine Learning. IEEE Access, 2020, (99):1-1, 124579-124607.

[3] Bashir, S., Maqbool, F., Khan, F.H. and Abid, A.S. Hybrid Machine Learning Model for Malware Analysis in Android Apps. Pervasive and Mobile Computing, 2023, 97, 101859. DOI:10.1016/j.pmcj.2023.101859.

[4] Mbunge, E., Muchemwa, B., Batani, J. and Mbuyisa, N. A review of deep learning models to detect malware in Android applications. Cyber Security and Applications, 2023, 1, 100014. DOI:10.1016/j.csa.2023.100014.

[5] Syrris V. and Geneiatakis, D. On machine learning effectiveness for malware detection in Android OS using static analysis data. Journal of information security and applications, 2021, 59, 102794. DOI:10.1016/j.jisa.2021.102794.

[6] Ogwara, N.O., Petrova, K. and Yang, M.L.B. MOBDroid2: An Improved Feature Selection Method for Detecting Malicious Applications in a Mobile Cloud Computing Environment. 2021 International Conference on Computational Science and Computational Intelligence (CSCI), 2021: IEEE, 397-403.

[7] McDonald J., Herron, N., Glisson, W. and Benton, R. Machine learning-based android malware detection using manifest permissions. Hawaii International Conference on System Sciences, 2021.

[8] Lee, J., Jang, H., Ha, S. and Yoon, Y. Android malware detection using machine learning with feature selection based on the genetic algorithm. Mathematics, 2021, 9(21), 2813. DOI: 10.3390/math9212813.

[9] Urooj, B., Shah, M.A., Maple, C., Abbasi, M.K. and Riasat, S. Malware detection: a framework for reverse engineered android applications through machine learning algorithms. IEEE Access, 10, 89031-89050, 2022.

[10] Fallah, S. and Bidgoly, A.J. Android malware detection using network traffic based on sequential deep learning models. Software: Practice and Experience, 2022, 52(9), 1987-2004. DOI: 10.1002/spe.3112.

[11] Oliveira, A.S. de and Sassi, R.J. Chimera: an android malware detection method based on multimodal deep learning and hybrid analysis. Authorea Preprints, 2020. DOI: 10.36227/techrxiv.13359767.

[12] Zhu, H.J. Gu, W., Wang, L.M., Xu, Z.C. and Sheng, V.S. Android malware detection based on multi-head squeeze-and-excitation residual network. Expert Systems with Applications, 2023, 212, 118705.

[13] Manzil, H.H.R. and Naik, S.M. Detection approaches for android malware: Taxonomy and review analysis. Expert Systems with Applications, 2023, 238(6), 122255. DOI: 10.1016/j.eswa.2023.122255.

[14] Sudar, K.M., Beulah, M., Deepalakshmi, P., Nagaraj, P. and Chinnasamy, P. Detection of Distributed Denial of Service Attacks in SDN using Machine learning techniques. International conference on Computer Communication and Informatics (ICCCI), 2021: IEEE, 1-5. DOI:10.1109/ICCCI50826.2021.9402517.

[15] Rathore, H., Nandanwar, A., Sahay, S.K. and Sewak, M. Adversarial superiority in android malware detection: Lessons from reinforcement learning based evasion attacks and defenses. Forensic Science International: Digital Investigation, 2023, 44, 301511. DOI:10.1016/j.fsidi.2023.301511.

[16] Dissanayake, S., Gunathunga, S., Jayanetti, D., Perera, K., Liyanapathirana, C. and Rupasinghe., L. An Analysis on Different Distance Measures in KNN with PCA for Android Malware Detection. International Conference on Advances in ICT for Emerging Regions, 2022: IEEE, 78-182. DOI: 10.1109/ICTer58063.2022.10024079.

[17] Strecker, S., Dave, R., Siddiqui, N. and Seliya, N. A modern analysis of aging machine learning based IOT cybersecurity methods. Computer Science, 2021. DOI:10.48550/arXiv.2110.07832.

[18] Mohamed, S.E., Ashaf, M., Ehab, A., Shereef, O., Metwaie, H. and Amer, E. Detecting malicious android applications based on API calls and permissions using machine learning algorithms. International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), 2021: IEEE, 1-6. DOI: 10.1109/MIUCC52538.2021.9447594.

[19] Salah, A.T., Hassan, M.A., Abbas, M.I., Sayed, Y.H., Elsahaer, Z.E. and Khoriba, G.A. Android Static Malware Detection using tree-based machine learning approaches. International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), 2022: IEEE, 3-10. DOI: 10.1109/MIUCC55081.2022.9781765.

[20] Mantoro, T., Stephen, D. and Wandy, W. Malware Detection with Obfuscation Techniques on Android Using Dynamic Analysis. International Conference on Computing, Engineering and Design (ICCED), 2022: IEEE, 1-6. DOI: 10.1109/ICCED56140.2022.10010359.

[21] Banik, A. and Singh, J.P. Android Malware Detection by Correlated Real Permission Couples Using FP Growth Algorithm and Neural Networks. 2023: IEEE Access, 11, 124996-125010. DOI: 10.1109/ACCESS.2023.3323845.

[22] Johnson, S., Donner, R. and Perez, A.J. Comparing Classifiers: A Look at Machine-Learning and the Detection of Mobile Malware in COVID-19 Android Mobile Applications. International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, 2023, 498-503. DOI: 10.1145/3565287.3617629.

[23] Gautam, S. Rathi, K., Bagwari, A. and Alkhayyat, A. Application of machine learning in malware detection for Android. AIP Conference Proceedings, 2023, 2930(1), AIP Publishing. DOI: 10.1063/5.0176731.

[24] Ibrahim, M. Abdullahi, A., Ahmad, A.M.A. and Mustapha, R. A Comparative Analysis of Android Malware Detection with and without Feature Selection Techniques using Machine Learning. 2023. DOI: 10.56471/slujst.v6i.371.

[25] Xie, N., Qin, Z. and Di, X. GA-StackingMD: Android Malware Detection Method Based on Genetic Algorithm Optimized Stacking. Applied Sciences, 2023, 13(4), 2629. DOI: 10.3390/app13042629.