INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING & COMPUTER SYSTEMS (IJSECS)
ISSN: 2289-8522 e-ISSN: 2180-0650
VOL. 9, ISSUE 2, 140 – 149
DOI: https://doi.org/10.15282/ijsecs.9.2.2023.7.0118

ORIGINAL ARTICLE

# PERFORMANCE ANALYSIS OF SELECTED CLASSIFICATION ALGORITHMS ON ANDROID MALWARE DETECTION

G.K. Afolabi-Yusuf[1], Y.O. Olatunde[1], K.Y. Obiwusi[1], M.O. Yusuf[2], O.C. Abikoye[2]

[1]Department of Mathematics and Computer Science, Summit University Offa, Kwara state, Nigeria.
[2]Department of Computer Science, University of Ilorin, Ilorin, Nigeria

**ABSTRACT** – Android mobile devices are widely used across all platforms and the development of malicious apps can compromise a user's mobile system. Considering the large amount of new malicious apps, there is a need for a detection system that can operate efficiently to identify these apps. The study analyzes and compares the performance of DREBIN and MALGENOME data sets with the dataset's SMOTE version on selected machine learning algorithms using WEKA tools. The performance of bayesian, function, rule, and tree-based classification algorithms on the two datasets was explored in this work. WEKA tool was used in pre-processing and SMOTE class balancing of the datasets before the model training using different classification algorithms on the two datasets and the performance evaluation. In the performance evaluation, parameters such as accuracy, precision, f-measure, the area under cover, true positive, recall, and false positive rate were employed. According to the study, tree-based classifiers (Recursive Tree, Decision Tree and Classification and Regression Tree) algorithms have 97.24%, 98.21% and 98.21% accuracy on the Malgenome dataset and 97.30%, 97.33% & 97.28% of accuracy on Drebin dataset and function-based classifiers (Support Vector Machine (SVM) and Logistic Regression) algorithms has 97.81% & 96.87% of accuracy on Malgenome dataset and 97.00% & 97.81% of accuracy on Drebin dataset which concludes that classifier algorithms in these groups proofed to be promising for the detection of android malware. The function-based classifier is the most outstanding method for the two datasets as it outperforms all other classifiers for both classes with 97.81% and 97.33%. SVM and Logistic Regression, are highly effective in detecting malicious Android apps, outperforming other classifier types with accuracy rates up to 97.81%. Tree-based classifiers also showed strong performance across DREBIN and MALGENOME datasets. This research underscores the potential of function-based algorithms as robust tools for enhancing mobile security against malware threats.

**ARTICLE HISTORY**
Received: 22 May 2023
Revised: 17 November 2023
Accepted: 11 March 2024
Published: 18 March 2024

**KEYWORDS**
*Cyber security*
*Malgenome*
*Drebin*
*Android*
*Malware Detection*

## INTRODUCTION

Android mobile device is widely used across all platforms in the world with more than 85% of the market share. Currently, millions of software applications are available for Android users on the device store and these apps record billions of downloads yearly [1]. Android device users can install software applications from any source which has unfortunately spurs the development of malicious apps that can compromise user's mobile systems [2].

Malicious software has been created incessantly over the years to compromise the security of the Android device platform. Android encryption and protection structure contribute immensely to the growth in the complexity of Android Malware [3]. Malware developers evolve with the new detection capabilities and are always trying to counteract the developers' security strengths. Malware always takes advantage of the least vulnerable and security flaws to infect [4]. Malware causes infliction on the affected mobile devices; it can render the phone unusable, account fraud and phishing of private information or eradicate the user's phone memory.

It has been discovered that there are millions of malicious Android apps available. These apps are designed to launch innumerable attacks using Trojan horses, worms, exploits, and viruses [5]. Because there are many different versions of these malicious apps installed, it might be challenging to identify, remove, or destroy them [6]. Researchers and analysts have developed a number of Android malware detection apps that employ machine learning algorithms and data mining approaches to detect and prevent malware in an effort to avert security breaches. Given the volume of newly released harmful apps, a detection system that is effective in identifying these apps is required. Researchers have developed several machine learning approaches, that learned from attributes extracted using static/dynamic approaches to identify malicious applications. However, such models suffer from low detection accuracy, due to the presence of noisy attributes, extracted from conventional feature selection algorithms [7].

In machine learning, the Synthetic Minority Over-sampling Technique (SMOTE) proves invaluable in mitigating the challenges posed by class imbalance in datasets, particularly prevalent in binary classification problems [5]. The imbalance arises when one class is significantly underrepresented compared to the other, a scenario common in real-world applications such as fraud detection or medical diagnosis. SMOTE addresses this by generating synthetic instances

*CORRESPONDING AUTHOR | G. K. Afolabi-Yusuf | ✉ afolabi.ganiyat@summituniversity.edu.ng

of the minority class, effectively rebalancing the dataset and preventing models from being biased towards the majority class [1]

By augmenting the minority class with synthetic examples, SMOTE enhances model performance by allowing it to better discern the characteristics of the minority class. This improvement translates into better generalization to unseen data, as models are no longer skewed towards the majority class. Moreover, SMOTE aids in preventing overfitting, a common pitfall in imbalanced datasets where models can become overly sensitive to the majority class. The technique achieves this by introducing diversity through synthetic instances, thereby fostering more nuanced decision boundaries and facilitating a more robust and accurate classification process.

The study analyzes and compares the performance of Bayesian, Function, Rule, and Tree-based classification algorithms on the detection of Android Malware using DREBIN, MALGENOME and the SMOTE version of the two datasets on selected metrics. This study is conducted using WEKA Tool and the classification algorithms used are Bayesian Based classifiers [Naïve Bayes (NB) and Bayesian Network(BN)], Function-Based classifiers [Support Vector Machine (SVM) and Logistic Regression (LR)], Rule-Based classifiers [Classification rule (CR) and Decision Table (Dtab)] and the Tree-Based classifiers [Recursive Tree (RT), Decision Tree (DT) and Classification and Regression Tree (CART)].

The remaining part of this paper is as presented herewith: A basic Review of related works is presented in Section II while the methods used in collecting the Android Malware datasets, analyzing the performance of the machine learning algorithms on the detection of the malware and evaluating the performance of the Machine Learning algorithm is discussed in Section III. In section IV, the result of the evaluation is presented and section V discusses the conclusion of this work.

## RELATED WORK

This section presents a review of related works conducted on Android malware detection approaches.

Due to its widespread use, Android is susceptible to a variety of harmful attacks [8]. Therefore, in order to mitigate this risk, malware detection is necessary. [4] studied how to use ensemble learning to detect Android malware more accurately. An ensemble model was created by combining the predictions of the Random Forest, Support Vector Machine, and K Nearest Neighbor algorithms as basis models with a majority vote combination function. With a sensitivity of 97.9% and a classification accuracy of 98.00%, Random Forest was found to have the highest performance among all classifier-based algorithms in the study. The ensemble model's classification accuracy was 98.16%, while its sensitivity was 98.16%. The study found that the most optimal detection model was the ensemble learner. [6] suggested a malware detection technique that concentrates on the URLs that the programs on the device visit. Their technique makes use of a multi-view neural network, which generates several input views and assigns soft attention weights to various input features. The views don't require complex feature engineering, preserving the semantic information from inputs for classification. The results of the experiment shown that the method was able to detect potentially harmful programs in the third-party app market as well as malware that was found in various months of a given year. It also achieved timely and robust malware detection.

While they employ distinct approaches, [4] and [6] both suggested strategies for identifying Android malware. [4] optimized Android malware detection by ensemble learning. In order to generate an ensemble model, the study created base models from three separate algorithms and aggregated their predictions using a majority vote combination function. High sensitivity and classification accuracy were attained by their method. Conversely, [6] suggested a technique that concentrates on the URLs that the programs on the device visit. The technique distributes soft attention weights to focus on distinct input aspects by using a multi-view neural network to automatically generate numerous perspectives of the inputs. The views preserve the semantic information from inputs for classification without requiring complicated feature engineering. The method also achieved robust and timely malware detection.

[6] have proposed a multi-view neural network approach that may be used to generate multiple views of the inputs (e.g., URLs visited by the applications on the device), and then combine the predictions of multiple base models created from different algorithms using an ensemble learning approach akin to [4]. It is important to remember, nevertheless, that in order to compare the efficacy of this strategy to the individual techniques put out by Christiana et al. (2020) and Wang et al. (2020), it would need to be evaluated experimentally.

There are certain shortcomings in the research done by [6] and [4]. [4] did not investigate different classifiers or feature selection strategies; instead, they solely employed three base classifiers to build the ensemble model. Consequently, it's possible that the findings cannot be applied to other feature selection techniques or classifiers. Furthermore, the study's dataset was neither particularly vast or diversified, which would limit how broadly the findings can be applied to other datasets with other features. Last but not least, there is no assessment of the ensemble model's performance in identifying zero-day or previously unidentified malware, which is a difficult undertaking in the realm of malware detection. However, [6] suggested a technique that might not be useful for identifying malware that doesn't rely on network connectivity or URLs that the device's applications visit. Since the study's dataset was small, it is uncertain

how well the strategy would work with larger and more varied datasets. The neural network used in the suggested method has a lot of parameters, thus in order to get the best results, it could need a lot of processing power and training data.

Users' privacy and file integrity are seriously threatened by malware [9]. In order to identify and halt dangerous behaviors, [9] introduce MADAM, a host-based malware detection system that correlates and analyzes features at four levels. Using two parallel classifiers and a behavioral signature-based detector, MADAM is demonstrated to prevent over 96% of malicious apps from three big datasets. This method's drawback is that it depends on behavioral signatures, which might not be able to detect every kind of malware.

But [10] unveiled SIGPID, a malware detection system that distinguishes between safe and dangerous programs based on permission usage analysis. Using machine learning-based classification techniques, SIGPID determines which permissions are most crucial for effectively distinguishing between the two. With 93.62% of the malware in the dataset and 91.4% of unknown or novel malware samples detected, the study demonstrates that SIGPID is more effective than alternative methods. This method's drawback is that it might miss malware that doesn't employ any dubious permissions.

Online Android Malware Detection (ONAMD), another method for Android malware detection proposed by [1], makes use of both static and dynamic features. Applications are categorized as benign or dangerous by ONAMD using SVM and Random Forest algorithms once it extracts the requested permissions. When compared to earlier deployments to Androguard, the method achieves a higher recall rate and is demonstrated to be efficient, consuming half the time. This method's drawback is that malware that employs obfuscation to avoid detection can slip through the cracks.

In general, the methods for Android malware detection proposed by [9], [10] and [1] yield differing degrees of performance. While SIGPID achieves excellent precision, recall, accuracy, and f-measure while requiring less analysis times, MADAM has a high detection rate and a low false alarm rate; ONAMD, on the other hand, is efficient and outperforms Androguard in terms of recall rate. All approaches have drawbacks, though, like SIGPID's dependency on permission usage, MADAM's reliance on behavioral signatures, and ONAMD's use of a restricted feature set.

[3] conducted an analysis of crucial prerequisites essential for the deployment of Android malware detection systems in real-world scenarios. The examination delineated requirements that entail testing candidate approaches against a continually evolving stream of data. The study identified the design and implementation of an ensemble approach for automated Android malware detection that aligns with real-world demands. The Atomic Naive Bayes classifiers utilized as inputs for the Support Vector Machine ensemble are based on distinct APK feature categories, ensuring rapid processing and enhanced reliability against potential attackers through diversification. The research encompasses the initial publicly accessible outcomes produced against evolving data streams, incorporating nearly 1 million samples, with a model trained on an extensive sample set comprising 120,000 samples.

The technique known as DREBIN was presented by [11] as a means of identifying Android malware that functions directly on the smartphone. It gathers application features via a wide static analysis and embeds them in a shared vector space to find patterns suggestive of malware. DREBIN surpassed other equivalent methods in an examination of over 123,000 apps and 5,500 malware samples, detecting 94% of malware with few false alarms. The technique reveals pertinent characteristics of the malware that has been found and offers reasons for each detection. DREBIN can be used to examine downloaded applications right on the device because it takes an average of 10 seconds to assess an application on common smartphones.

In conclusion, Android is vulnerable to malicious attacks and the detection of malware is crucial. Different approaches have been proposed for Android malware detection, including ensemble learning, multi-view neural networks, host-based malware detection, and permission-based analysis. These methods have varying levels of effectiveness and limitations. Ensemble learning and multi-view neural networks show promising results, but further evaluation is needed to determine their effectiveness compared to other methods. Host-based malware detection and permission-based analysis also have their strengths and limitations. To effectively detect Android malware, it may be necessary to combine different approaches and techniques to enhance the accuracy and effectiveness of the detection system.

It is deduced from the review of related works that several approaches have been developed for the detection of Android malware such as MADAM, SIGPID, and ONAMD using existing and new datasets such as the NSL-KDD, KDDCup1999, and DREBIN among other datasets. These approaches enabled the use of machine learning approaches such as the Support Vector Machine SVM, Rough Set Theory, Random Forest, and K-Nearest Neighbor for enhancing the detection. The performance of the approaches was evaluated based on some metrics such as the False Positive Rate, Accuracy, and Average Cost of Misclassification and the results reported give high performance in terms of the metrics.

## METHODOLOGIES

The performance analysis and evaluation approach is based on data collection, data pre-processing, SMOTE data balancing, feature classification, model development, and model performance evaluation using selected data analysis metrics and Performance Comparison as presented in Figure 1.

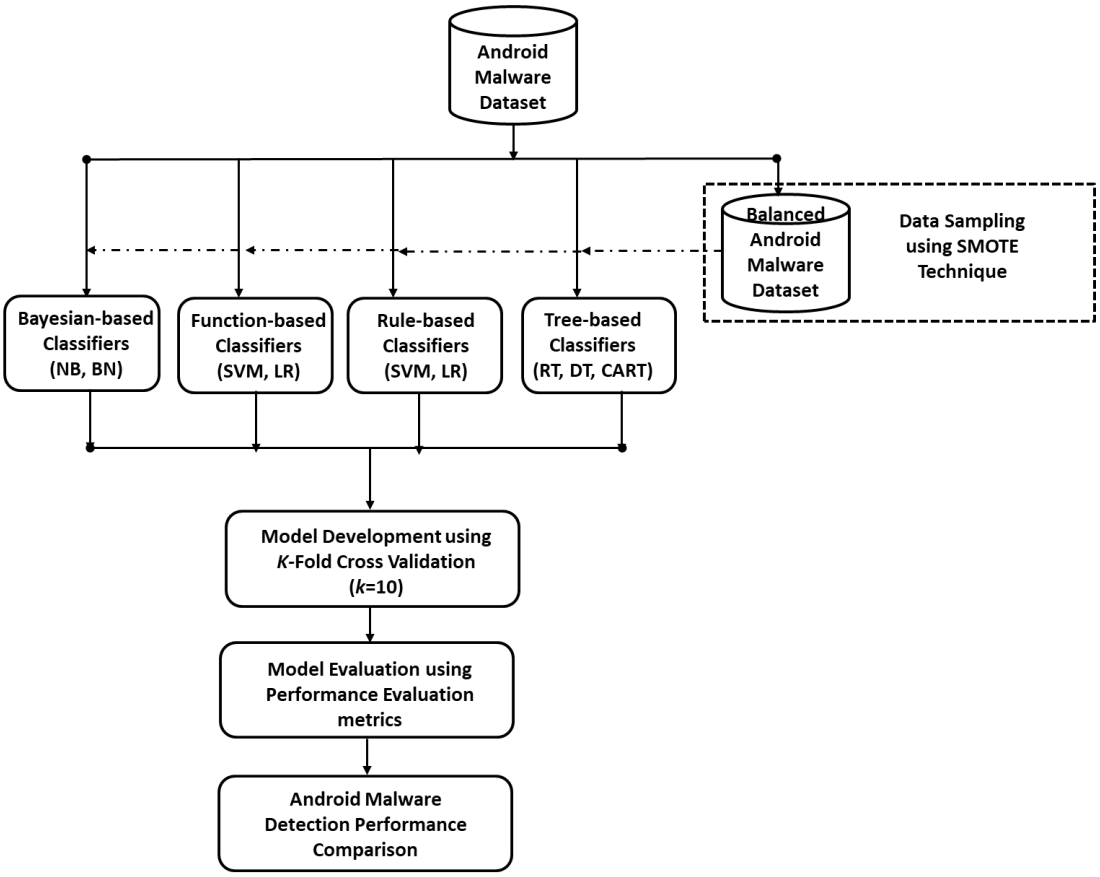


**Figure 1: The Study Experiment Framework and Approach**

### Dataset Collection and Description

The first process in this study as presented in Figure 1 is data collection. The MALGENOME and DREBIN datasets used in this study were obtained from the UCI repository

DREBIN dataset consists of approximately 123,453 Android applications (APK files) that were collected from various sources such as the Google Play Store, third-party app stores, and online repositories. Each APK file in the dataset is labelled as either benign or malicious. The malicious applications were collected from different malware databases and websites, and they were manually analyzed by security experts to confirm their malicious behaviour. The benign applications were collected from various sources, and they do not contain any malicious behaviour. The dataset provides various information about each application such as permissions, API calls, intents, and other features that can be used to analyze and classify them. The dataset also includes metadata such as the application name, version, and description, as well as the SHA256 hash value of each APK file for easy identification and tracking.

The DREBIN dataset has been widely used by researchers to develop and evaluate different Android malware detection methods.

MALGENOME is a publicly available dataset of Android malware, which was released in 2017. It contains 1,260 malware samples and 1,200 benign applications (i.e., non-malicious apps) that were collected between 2010 and 2012 from various sources, including the android market (now known as google play store), third-party markets, and various other websites. Each android app in the dataset is provided in the form of an APK (Android Package) file, which contains the compiled code and resources of the app. The dataset also includes a set of permissions that are requested by each app, along with information about the package name, version, and file size.

Overall, the MALGENOME dataset is a valuable resource for the research community

Summarily, the Malgenome dataset has 1260 applications grouped into 49 malware families and the Drebin dataset has a total of 5560 applications grouped into 179 malware families.

### Data Pre-Processing

The data pre-processing was conducted by removing the noisy and inconsistent data from the dataset to extract a suitable format required for the detection of Android malware using selected classification algorithms. Then, the values in the input features of the datasets were scaled using a min-max scalar to improve the performances of the proposed learning algorithms. Also, the SMOTE version of the dataset was carried out to remove class imbalance.

Thus, two sets of each dataset were used in this study: the cleaned raw data and the SMOTE data.

The SMOTE's flow of control is attached herewith in Figure 2:

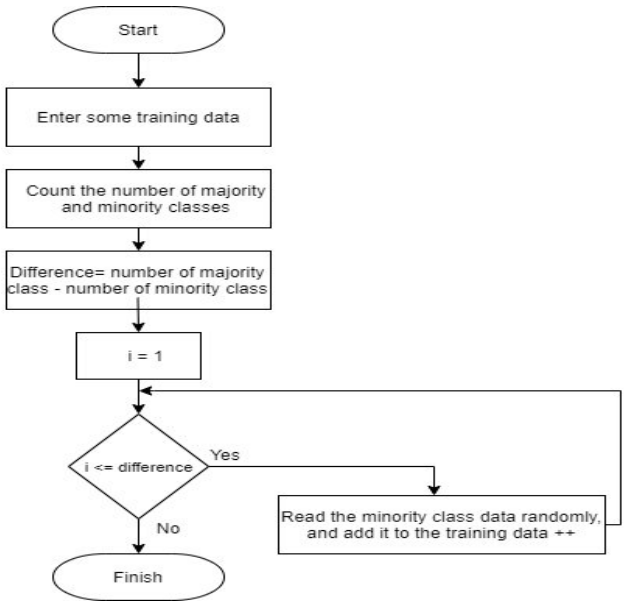


**Figure 2**: Synthetic Minority Oversampling Technique Flowchart (Sopiyan et al., 2022)

### Feature Classification and Model Development

The k-fold cross-validation model was developed using WEKA. Attached below is the pseudocode used for each of the datasets.

The pseudocode is as follows:

1. Start
2. Load the Dataset
3. Choose Classifier
4. Set Cross Validation Options to 10 folds
5. Run Cross Validation
6. Examine various evaluation metrics (accuracy, precision, recall, etc.) in the output
7. Repeat for Each Classifier
8. Save the cross-validation results for further analysis.
9. End.

In the process of developing a k-fold cross-validation model using Weka, the initial step involves loading the dataset within the Weka Explorer interface. Subsequently, the choice of a classifier is made from options such as NaiveBayes, J48 for Decision Trees, or SMO for SVM, and specific configurations are set using the "More options..." button. The subsequent step involves configuring k-fold cross-validation by navigating to the designated tab and specifying the desired number of folds, commonly set to 10. Initiating the model evaluation, the user clicks the "Start" button, prompting Weka to execute the k-fold cross-validation process. Following completion, the results are thoroughly analyzed, encompassing metrics like accuracy, precision, and recall, with additional visualizations and summary statistics provided by Weka for comprehensive assessment. This iterative process was repeated for all the classifiers used and the cross-validation results are saved for further examination or comparison.

### Performance Evaluation

The effect of class imbalance on Machine Learning classification algorithms used in Android malware detection and its SMOTE version was evaluated and analyzed using WEKA tools. Several evaluation metrics were used for the study and are thus described herewith:

i. Accuracy: Accuracy is a fundamental metric used in classification to assess the overall correctness of a model's predictions across all classes. It is the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances in the dataset. Mathematically, accuracy is calculated using the formula:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{1}$$

ii. Area Under the Curve (AUC) is a metric commonly used in evaluating the performance of binary classification models, particularly in the context of receiver operating characteristic (ROC) curves. ROC curves illustrate the trade-off between a true positive rate (sensitivity) and a false positive rate (1 - specificity) at various thresholds.

AUC specifically quantifies the area under the ROC curve, providing a single value that represents the model's ability to distinguish between positive and negative instances across different threshold settings. A higher AUC indicates better discrimination, with a perfect classifier achieving an AUC of 1.

The AUC value ranges from 0 to 1, where 0.5 suggests that the model performs no better than random chance. AUC is advantageous because it is threshold-independent, meaning it considers the model's overall ability to discriminate without being sensitive to a specific decision threshold.

iii. Fmeasure: F-measure, also known as the F1 score, is a metric used to evaluate the performance of a classification model, particularly in the context of binary classification. It combines precision and recall into a single value, providing a balanced assessment of a model's effectiveness. The F-measure is the harmonic mean of precision and recall, giving equal weight to both. It is calculated using the formula in Equation 2:

$$\textbf{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

The F-measure ranges from 0 to 1, with 1 indicating perfect precision and recall. It's a valuable metric in scenarios where achieving a balance between false positives and false negatives is crucial, such as medical diagnoses or spam detection.

iv. Precision: Precision is a metric used in binary classification to evaluate the accuracy of positive predictions made by a model. It is the ratio of true positive predictions to the total predicted positive instances, indicating the proportion of correctly identified positive cases among all instances predicted as positive.

In simpler terms, precision answers the question: "of all the instances predicted as positive, how many were actually positive?" A higher precision value indicates fewer false positives and, therefore, a more accurate positive prediction by the model. Precision is especially important when the cost of false positives is high.

Mathematically, precision is calculated using the formula in equation 3:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{3}$$

v. Recall: Recall, also known as sensitivity or true positive rate, is a metric used in binary classification to evaluate a model's ability to correctly identify all instances of a positive class. It is the ratio of true positive predictions to the total actual positive instances.

laMathematically, recall is calculated using the formula in equation 4:

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

In simpler terms, recall answers the question: "Of all the actual positive instances, how many were correctly identified by the model?" A higher recall value indicates that the model is capturing a larger proportion of the actual positive instances. Recall is particularly important in scenarios where missing positive instances (false negatives) is a critical error, even if it results in more false positives.

vi. False Positive Rate (FPR): is a key metric in binary classification that measures the proportion of actual negative instances that are incorrectly classified as positive by a model, out of the total number of true negatives and false positives. FPR is calculated using the formula in equation 5:

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

In essence, FPR quantifies the model's tendency to produce false alarms or Type I errors, indicating the rate at which negative instances are mistakenly identified as positive.

## RESULTS DISCUSSION

The Android malware dataset collected was pre-processed and cleaned. Then the dataset's features were selected for the model-building stage which involves the detection of malware using the selected classification algorithms. The performance of each of the algorithm's models is evaluated using selected metrics and the results are analyzed and presented in Table 1, 2, 3 and 4 attached.

The experimental analysis results are discussed in this section.

To conduct the performance comparison of the different classification algorithms on the processed datasets and SMOTE datasets, Bayesian-based (NB & BN), Function-based (SVM & LR), Rule-based (Dtab & RT), and Tree-based (DT & CART) classifiers were used. The classifiers were used separately for the classification of the features used for the training of the K-fold model. The model performance is then evaluated using selected metrics such as accuracy, AUC, fmeasure, precision, recall, TPR and FPR.

The performance evaluation outcomes for each classifier are detailed in Tables 1, 2, 3, and 4, organized according to the specific datasets in use. These tables comprehensively present the classifier performance across various metrics, encompassing accuracy, precision, f-measure, area under the curve, true positive, recall, and false positive rate.

**Table 1**. Performance Analysis of Malgenome Dataset

| Classifiers | Accuracy | AUC | F-measure | Precision | Recall | TPR | FPR | |
|---|---|---|---|---|---|---|---|---|
| NB | 92.58 | 0.992 | 0.927 | 0.935 | 0.926 | 0.926 | 0.048 | Bayesian-based Classifiers |
| BN | 92.73 | 0.992 | 0.929 | 0.937 | 0.927 | 0.927 | 0.046 | |
| SVM | 97.81 | 0.969 | 0.978 | 0.978 | 0.978 | 0.978 | 0.04 | Function-based Classifiers |
| LR | 96.87 | 0.984 | 0.969 | 0.969 | 0.969 | 0.969 | 0.028 | |
| CR | 75.44 | 0.809 | 0.761 | 0.843 | 0.754 | 0.754 | 0.139 | Rule-based Classifiers |
| Dtab | 93.81 | 0.977 | 0.938 | 0.939 | 0.938 | 0.938 | 0.066 | |
| RT | 97.24 | 0.97 | 0.972 | 0.972 | 0.972 | 0.972 | 0.033 | Tree-based Classifier |
| DT | 98.21 | 0.983 | 0.982 | 0.982 | 0.982 | 0.982 | 0.022 | |
| CART | 98.21 | 0.987 | 0.982 | 0.982 | 0.982 | 0.982 | 0.023 | |

Table 1 presents the performance analysis of selected machine learning classification algorithms on the Malgenome Dataset based on selected metrics.

The performance of the classifiers was evaluated on distinct metrics, showcasing the effectiveness of each algorithm on the Malgenome datasets. In the Bayesian-based category, Naïve Bayes (NB) and Bayesian Network (BN)

demonstrated robust accuracy, AUC, and precision, with NB achieving 92.58% accuracy and BN slightly surpassing at 92.73%. Function-based classifiers, including Support Vector Machine (SVM) and Logistic Regression (LR), exhibited high accuracy of 97.81% and 96.87%, respectively, emphasizing their efficacy in classification tasks. Rule-based classifiers, represented by Classification Rule (CR), Decision Table (Dtab), and Recursive Tree (RT), displayed varying performance, with CR achieving 75.44% accuracy, Dtab demonstrating 93.81%, and RT showcasing 97.24%. Lastly, Tree-based classifiers, encompassing Decision Tree (DT) and Classification and Regression Tree (CART), excelled with 98.21% accuracy, reinforcing their reliability in classification scenarios.

These results could be construed as evidence that function-based and tree-based classifiers outperformed others in terms of accuracy. Their ability to achieve higher accuracy rates across the evaluated metrics, including AUC, F-measure, Precision, Recall, TPR, and FPR, suggests that these classifiers are effective.

In summary, the evidence from Table 1 shows that function-based and tree-based classifiers showcased superior performance in the classification task compared to Bayesian-based and rule-based classifiers.

**Table 2**: Performance Analysis of DREBIN Dataset

| Classifiers | Accuracy | AUC | F-measure | Precision | Recall | TPR | FPR | |
|---|---|---|---|---|---|---|---|---|
| NB | 82.42 | 0.928 | 0.827 | 0.861 | 0.824 | 0.824 | 0.125 | Bayesian-based Classifiers |
| BN | 82.78 | 0.928 | 0.831 | 0.862 | 0.828 | 0.828 | 0.124 | |
| SVM | 97.00 | 0.964 | 0.970 | 0.970 | 0.970 | 0.970 | 0.043 | Function-based Classifiers |
| LR | 97.81 | 0.995 | 0.978 | 0.978 | 0.978 | 0.978 | 0.027 | |
| CR | 75.55 | 0.798 | 0.758 | 0.833 | 0.756 | 0.756 | 0.159 | Rule-based Classifiers |
| Dtab | 92.20 | 0.976 | 0.922 | 0.924 | 0.922 | 0.922 | 0.077 | |
| RT | 97.30 | 0.972 | 0.973 | 0.973 | 0.973 | 0.973 | 0.030 | Tree-based Classifier |
| DT | 97.33 | 0.975 | 0.973 | 0.973 | 0.973 | 0.973 | 0.032 | |
| CART | 97.28 | 0.98 | 0.973 | 0.973 | 0.973 | 0.973 | 0.033 | |

Table 2 presents the performance analysis of selected Machine Learning classification algorithms on the Drebin dataset based on selected metrics.

In Table 2, the performance of the classifiers on the evaluated metrics is demonstrated. The Bayesian-based classifiers, Naïve Bayes (NB) and Bayesian Network (BN) both exhibited commendable accuracy, AUC, and precision, with NB achieving 82.42% accuracy and BN slightly exceeding 82.78%. In the function-based category, Support Vector Machine (SVM) and Logistic Regression (LR) showcased robust performance, particularly LR, which achieved the highest accuracy at 97.81% and an impressive AUC of 0.995. Rule-based classifiers, represented by Classification Rule (CR), Decision Table (Dtab), and Recursive Tree (RT), displayed varying performance, with CR achieving 75.55% accuracy, Dtab demonstrating 92.20%, and RT showcasing 97.30%. Within the tree-based classifiers, both Decision Tree (DT) and Classification and Regression Tree (CART) demonstrated strong accuracy at 97.33% and 97.28%, respectively. These results provide valuable insights into the comparative strengths of each classifier, emphasizing the high performance of function-based and tree-based classifiers in this specific dataset. It is thus, evident in Table 2 that Function-Based Classifiers outperform other Classifiers in terms of Accuracy.

**Table 3**: Performance Analysis of SMOTE Version of Malgenome Dataset

| Classifiers | Accuracy | AUC | F-measure | Precision | Recall | TPR | FPR | |
|---|---|---|---|---|---|---|---|---|
| NB | 95.26 | 0.994 | 0.953 | 0.954 | 0.954 | 0.953 | 0.047 | Bayesian-based Classifiers |
| BN | 94.5 | 0.994 | 0.945 | 0.948 | 0.945 | 0.945 | 0.055 | |
| SVM | 98.24 | 0.982 | 0.982 | 0.983 | 0.982 | 0.982 | 0.018 | Function-based Classifiers |
| LR | 99.33 | 0.998 | 0.993 | 0.993 | 0.993 | 0.993 | 0.007 | |
| CR | 81.42 | 0.809 | 0.809 | 0.857 | 0.814 | 0.814 | 0.185 | Rule-based Classifiers |
| Dtab | 93.64 | 0.986 | 0.936 | 0.937 | 0.936 | 0.936 | 0.063 | |
| RT | 97.9 | 0.979 | 0.979 | 0.979 | 0.979 | 0.979 | 0.021 | Tree-based Classifier |
| DT | 98.06 | 0.983 | 0.981 | 0.981 | 0.981 | 0.981 | 0.019 | |
| CART | 97.63 | 0.98 | 0.976 | 0.976 | 0.976 | 0.976 | 0.024 | |

Table 3 presents the performance analysis of selected machine learning classification algorithms on the SMOTE version of the Malgenome dataset based on selected metrics.

In Table 3, the classification performance of the classifiers is showcased. The bayesian-based category, Naïve Bayes (NB) and Bayesian Network (BN) demonstrated strong accuracy, AUC, and precision, with NB achieving 95.26% accuracy and BN closely following at 94.5%. Function-based classifiers, including Support Vector Machine (SVM) and Logistic Regression (LR), exhibited exceptional accuracy, particularly LR, which reached an impressive 99.33% accuracy and an AUC of 0.998. Rule-based classifiers, represented by Classification Rule (CR), Decision Table (Dtab), and Recursive Tree (RT), displayed varying performance, with CR achieving 81.42% accuracy, Dtab demonstrating 93.64%, and RT showcasing 97.9%. Among the tree-based classifiers, both Decision Tree (DT) and Classification and Regression Tree (CART) achieved high accuracy at 98.06% and 97.63%, respectively.

These results underscore the robust performance of function-based and tree-based classifiers across the evaluated metrics in this specific SMOTE version of the Malgenome dataset.

**Table 4**: Performance Analysis of SMOTE Version of Drebin Dataset

| Classifiers | Accuracy | AUC | F-measure | Precision | Recall | TPR | FPR | |
|---|---|---|---|---|---|---|---|---|
| NB | 85.61 | 0.945 | 0.855 | 0.869 | 0.856 | 0.856 | 0.144 | Bayesian-based Classifiers |
| BN | 85.58 | 0.942 | 0.855 | 0.869 | 0.856 | 0.856 | 0.144 | |
| SVM | 97.24 | 0.972 | 0.972 | 0.973 | 0.972 | 0.972 | 0.028 | Function-based Classifiers |
| LR | 98.10 | 0.996 | 0.981 | 0.981 | 0.981 | 0.981 | 0.019 | |
| CR | 80.10 | 0.798 | 0.795 | 0.840 | 0.801 | 0.801 | 0.199 | Rule-based Classifiers |
| Dtab | 92.41 | 0.976 | 0.924 | 0.924 | 0.924 | 0.924 | 0.076 | |
| RT | 97.74 | 0.978 | 0.977 | 0.977 | 0.977 | 0.977 | 0.023 | Tree-based Classifier |
| DT | 97.77 | 0.982 | 0.978 | 0.978 | 0.978 | 0.978 | 0.022 | |
| CART | 97.65 | 0.984 | 0.976 | 0.977 | 0.976 | 0.960 | 0.024 | |

Table 4 presents the performance analysis of selected Machine Learning classification algorithms on the SMOTE version of the Drebin dataset based on selected metrics.

In Table 4, the performance of the classifiers is detailed. The Bayesian-based classifiers, Naïve Bayes (NB) and Bayesian Network (BN) exhibited similar accuracy and AUC, both achieving around 85.6% accuracy. Function-based classifiers, represented by Support Vector Machine (SVM) and Logistic Regression (LR), demonstrated high accuracy, particularly LR, which achieved 98.1% accuracy and an AUC of 0.996. Rule-based classifiers, including Classification Rule (CR), Decision Table (Dtab), and Recursive Tree (RT), displayed varying performance, with CR achieving 80.1% accuracy, Dtab demonstrating 92.41%, and RT showcasing 97.74%. Among the tree-based classifiers, both Decision Tree (DT) and Classification and Regression Tree (CART) achieved accuracy above 97.6%. These results underscore the robust performance of function-based and tree-based classifiers across the evaluated metrics in this specific SMOTE version of the Drebin dataset.

In all of the measures used for this evaluation, the function-based classifier outperforms all other classifiers as depicted in Table 1, 2, 3 & 4.

## CONCLUSION

The performance of Bayesian, Function, Rule, and Tree-based algorithms on publicly accessible benchmark datasets (MALGENOME and DREBIN) is explored in this work, and the SMOTE version of the datasets is created to enhance the evaluation. The four datasets were pre-processed in the same way and used to train and evaluate the specified supervised learning algorithms. In the performance evaluation, parameters such as accuracy, precision, f-measure, the area under cover, true positive, recall, and false positive rate were employed. According to the study, both tree-based classifiers and function-based classifier algorithms have the most promising results in all of the metrics utilized for assessment, even though the function-based classifier is the most outstanding method for the two datasets. The study also concluded that depending on two datasets used for assessment, any of the function-based classifier algorithms can efficiently detect and identify malicious apps on android mobile devices.

## ACKNOWLEDGEMENT

The author would like to thank Universiti Malaysia Pahang Al-Sultan Abdullah (UMPSA) for accepting this article

## REFERENCES


[1] R. Riasat, M. Sakeena, A. H. Sadiq, and Y. J. Wang, "Onamd: An Online Android Malware Detection Approach," *Proc. - Int. Conf. Mach. Learn. Cybern.*, vol. 1, no. July, pp. 190–196, 2018, doi: 10.1109/ICMLC.2018.8526997.

[2] O. C. Abikoye, U. A. Ojo, J. B. Awotunde, and R. O. Ogundokun, "A safe and secured iris template using steganography and cryptography," pp. 23483–23506, 2020.

[3] P. Palumbo, L. Sayfullina, D. Komashinskiy, E. Eirola, and J. Karhunen, "A pragmatic android malware detection procedure," *Comput. Secur.*, vol. 70, pp. 689–701, 2017, doi: 10.1016/j.cose.2017.07.013.

[4] A. O. Christiana, B. A. Gyunka, and A. N. Oluwatobi, "Optimizing android malware detection via ensemble learning," *Int. J. Interact. Mob. Technol.*, vol. 14, no. 9, pp. 61–78, 2020, doi: 10.3991/ijim.v14i09.11548.

[5] O. C. Abikoye and B. Gyunka, "The Threat of Split-Personality Android Malware on Developing Economy School of Computing , Engineering & Physical Sciences Computing and Information Systems Journal Edited by Abel Usoro," no. February, 2018.

[6] S. Wang *et al.*, "Deep and broad URL feature mining for android malware detection," *Inf. Sci. (Ny).*, vol. 513, pp. 600–613, 2020, doi: 10.1016/j.ins.2019.11.008.

[7] A. Priya, S. Garg, and N. P. Tigga, "Predicting Anxiety, Depression and Stress in Modern Life using Machine Learning Algorithms," *Procedia Comput. Sci.*, vol. 167, no. September, pp. 1258–1267, 2020, doi: 10.1016/j.procs.2020.03.442.

[8] R. Singh and A. Gehlot, "Review on Intrusion Detection in Edge Based IOT," *2022 Int. Interdiscip. Humanit. Conf. Sustain.*, pp. 788–793, 2022, doi: 10.1109/IIHC55949.2022.10060587.

[9] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," *IEEE Trans. Dependable Secur. Comput.*, vol. 15, no. 1, pp. 83–97, 2018, doi: 10.1109/TDSC.2016.2536605.

[10] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018, doi: 10.1109/TII.2017.2789219.

[11] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," no. February, pp. 23–26, 2014, doi: 10.14722/ndss.2014.23247.