Performance Analysis of Machine Learning Methods with Class Imbalance Problem in Android Malware Detection

https://doi.org/10.3991/ijim.v16i10.29687

Abimbola Ganiyat Akintola[1], Abdullateef Oluwagbemiga Balogun[1,2(✉)], Hammed Adeleye Mojeed[1,3], Fatima Enehezei Usman-Hamza[1], Shakirat Aderonke Salihu[1], Kayode Sakariyau Adewole[1], Ghaniyyat Bolanle Balogun[1], Peter Ogirima Sadiku[1]
[1]Department of Computer Science, University of Ilorin, Ilorin, Nigeria
[2]Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Perak, Malaysia
[3]Institute of Technical Informatics and Telecommunication, Gdansk University of Technology, Gdańsk, Poland
Balogun.ao1@unilorin.edu.ng

Abstract—Due to the exponential rise of mobile technology, a slew of new mobile security concerns has surfaced recently. To address the hazards connected with malware, many approaches have been developed. Signature-based detection is the most widely used approach for detecting Android malware. This approach has the disadvantage of being unable to identify unknown malware. As a result of this issue, machine learning (ML) for detecting malware apps was created. Conventional ML methods are concerned with increasing classification accuracy. However, the standard classification method performs poorly in recognizing malware applications due to the unbalanced real-world datasets. In this study, an empirical analysis of the detection performance of ML methods in the presence of class imbalance is conducted. Specifically, eleven (11) ML methods with diverse computational complexities were investigated. Also, the synthetic minority oversampling technique (SMOTE) and random undersampling (RUS) are deployed to address the class imbalance in the Android malware datasets. The experimented ML methods are tested using the Malgenome and Drebin Android malware datasets that contain features gathered from both static and dynamic malware approaches. According to the experimental findings, the performance of each experimented ML method varies across the datasets. Moreover, the presence of class imbalance deteriorated the performance of the ML methods as their performances were amplified with the deployment of data sampling methods (SMOTE and RUS) used to alleviate the class imbalance problem. Besides, ML models with SMOTE technique are superior to ML models based on the RUS method. It is therefore recommended to address the inherent class imbalance problem in Android Malware detection.

Keywords—Android, malware detection, machine learning, data sampling

1 Introduction

In recent years, virtually every member of society uses the internet in their day-to-day activities either for social interaction, getting the latest information, health-related transactions, or for educational purposes [1, 2]. The increase in the number of internet usage has led to a rise in the growth and popularity of mobile devices such as smartphones or tablets, as well as the Android operating system (OS) that is mostly used on these devices. In particular, the Android OS has emerged as the top mobile OS with a significant global market value and presence. Records have shown that more than a billion Android devices have been purchased, with Google Play alone accounting for an estimated 65 billion mobile software application downloads [3, 4]. Consequentially, Android's increasing popularity and usage, as well as the development of third-party app stores, has rendered it vulnerable to a variety of malware [5]. Malware is a malicious software application developed to execute harmful payloads on victim devices such as computers, smartphones, etc. [3]. In other words, it can be referred to as any software application that performs unwanted and suspicious activities on target devices. Malware can be categorized into various types such as virus, worm, Trojan, rootkit, ransomware, etc. Malware variants can successfully steal confidential data, initialize distributed denial of service (DDoS) attacks, and perform disruptive damage to host systems. All these malware have the potential of being harmful to mobile devices and this has become a challenge in the field of information security [6, 7]. According to McAfee Labs Threats Report, the number of mobile malware is continually growing [5]. In a similar report, Kaspersky [8] detected more than 5 million malicious installation packages, which includes new variants of trojans and ransomware. As viable solutions, different techniques such as malware detection, vulnerability detection, and application reinforcement, to impose protection on the Android OS have been proposed and developed [9–11]. Malware detection is widely adopted among the proposed security protection measures to prevent dangerous applications. Primarily, malware detection methods can be divided into two types: signature-based and anomaly-based methods. The signature-based method relies on a collection of specified characteristics of such threats to identify malicious behaviour. However, while this method may reliably detect previously known malware, it cannot discover new unknown dangerous behaviours. That is, the signature-based malware detection method cannot detect zero-day attacks [12, 13].

The anomaly-based method, on the other hand, aims to detect malicious behaviour in the network by continually measuring any deviations from known typical behaviour. Since anomaly-based methods do not require prior knowledge of malware, they are more effective at identifying previously undiscovered malware. In comparison with conventional (i.e., statistical and knowledge-based) strategies, the performance of malware detection algorithms based on machine learning (ML) approaches outperforms the conventional methods [10, 14–16]. Studies have shown that security experts and researchers are now focusing on ML solutions for malware detection. Deployment of ML methods for malware detection involves the extraction of features from both malicious and benign Android applications and thereafter the generated dataset will be used to train ML methods and generate malware detection models. In particular, the features

are extracted using static and dynamic malware analysis. Static malware analysis investigates the Android application's code to identify the malware pattern without executing the code [6]. It offers high efficiency in identifying the apps. The shortcoming of this method is that it fails against obfuscation techniques [1]. The dynamic malware analysis examines the app's behaviour while running in a virtual environment like a sandbox. This method is effective but consumes more resources, time and is unable to explore all execution paths. The integration of both these methods gives better results using ML methods. However, the use of clean and well-defined datasets is critical when developing an ML-based malware model, because the performance of the ML model depends largely on the quality of the dataset [17, 18]. Specifically, the distribution of class labels in a dataset is critical for developing effective ML models. In real-world circumstances, the class label distribution is unequal and, in many cases, highly skewed. This inherent phenomenon is known as the class imbalance problem. In other words, the class imbalance problem occurs when one of the class labels in a dataset has many instances (majority class) and the other class has a limited number of instances (minority class). Inadequately balanced class labels in a dataset make the classification process more difficult and undependable for ML models [19–21].

As there are more instances of benign applications as compared to malicious applications, the Android malware detection can be said to have a class imbalance problem [1, 11, 12]. In this research, an extensive comparative performance analysis is conducted to ascertain the performance of ML methods in the presence of a class imbalance problem. Specifically, eleven (11) ML methods with diverse and distinct computational characteristics are deployed on imbalanced and balanced Android malware datasets. The synthetic minority over-sampling technique (SMOTE) and random under-sampling (RUS) is used as a data sampling method to balance the Android malware datasets. The primary aim of this research is to empirically evaluate and validate the detection performances of ML models with an immensely imbalanced dataset.

The main contribution of this research is summarized as follows:

i. A detailed empirical analysis of the performance of eleven (11) ML methods with diverse computational characteristics on balanced and imbalanced Android malware detection datasets.
ii. Investigation on the effect of data sampling on the performance of ML methods in Android malware detection.

The remaining part of these papers are arranged as follows: Section 2 discusses the related concepts and related works while section 3 illustrates the methodology for the analysis. Section 4 entails the presentation and discussion of experimental results and findings while Section 5 concludes this study.

2 Related works

The task of malware detection in android devices and applications using ML algorithms have been widely researched in the literature. Many earlier works in this domain have utilized baseline classifiers to detect Android malware. SafeDroid presented by

Sen, et al. [22] utilized Decision Tree (DT) based on structural features rather than application program interface (API) calls and permissions as commonly used by other approaches. Experimental results indicated that the approach with structural features could detect new malware better than API-based features. Application of other baseline ML algorithms can be found in [23–29].

Considerable efforts have also been made to enhance the performance of the baseline classifiers using ensemble approaches. Rahman and Saha [30] presented StackDroid- a 2-level architecture for detecting malware in Android devices based on stacked generalization to reduce the error rate. The first level is composed of baseline classifiers such as Extremely Randomized Tree (ERT), Random Forest (RF), Multi-Layer Perceptron (MLP) and Stochastic Gradient Descent (SGD). The second level utilized a meta-estimator/predictor Extreme Gradient Boosting (EGB) as the final predictor by stacking the initial predictions of the baseline classifiers. Evaluated on publicly available DREBIN dataset, StackDroid produced very promising results achieving up to 97% area under the curve (AUC) value and only 1.67 false-positive rates (FPR). Yerima and Sezer [3] presented DroidFusion a framework for detecting malware in mobile Android devices based on the fusion selected baseline classifiers prediction using various classifier rank aggregation algorithms. The performance was found to be superior to the stacked generalization approaches previously used in multilevel architecture systems.

Recently, Dhalaria and Gandotra [31] proposed a Cost-Sensitive Forest (CS-Forest) as a technique to combat the data imbalance problem in android malware detection. The method is composed of a group of decision trees that employ a cost-sensitive voting method to aggregate the individual predictions of the decision trees. Its performance was compared with similar approaches without cost-sensitive methods, it proved to be more effective in performance.

There have also been works that focused on comparative performance analysis of techniques in detecting malware in Android devices. Rana and Sung [32] compared the performance of individual classifiers: support vector machine (SVM), Neural networks (NN), naïve bayes (NB), DT, Linear Discriminant Analysis (LDA) and k Nearest Neighbor (KNN) in detecting Android malware. KNN was found to perform better than the rest. Salihu, et al. [33] evaluated the performance of four individual classifiers: SVM, K-means, NB, and DT in detecting Android malware. Their study supports the claim that ML algorithms are effective in detecting malware in Android devices. The performance of three individual classifiers RF, SVM, NB were evaluated by Agrawal and Trivedi [14] for Android malware detection. Though all the selected classifiers performed effectively, RF was found to be superior to all the selected classifiers. Shar, et al. [34] carried out an extensive experimental comparative study of selected classifiers from different classes of learning techniques. The study considered four statistical approaches, three rule induction approaches and three deep learning approaches. The evaluation involved both static analysis and dynamic analysis with features granularity based on API calls at class level and the sequence of API calls at method level. In all, the RF classifier trained with the static API sequence-based features achieved the best results. In Gyunka, et al. [35], performance comparison of six ML algorithms: NB, Logistics Regression (LR), RF, Classification And Regression Tree (CART), KNN, and

SVM leveraging on the permission-based feature set for anomaly Android malware detection. RF and KNN outperformed other algorithms considering Android permission features in malware detection. Various ensemble approaches were experimentally evaluated by Agrawal and Trivedi [36] for the detection of malware in Android devices. Out of all the nine ensemble techniques considered, the Category Boosting (CatBoost) algorithm outperformed the rest.

Most of the existing comparative performance analyses conducted did not consider the class imbalance problems in their investigation. This work is orthogonal to previous performance comparative analysis as it considered evaluation of ML algorithms with and without class imbalance problem in Android malware detection.

3 Methodology

This section describes the research approach that was used in this study. Specifically, details on the implemented classifiers are presented. Also, the experimented Android malware datasets, performance evaluation metrics and the experimental procedure are illustrated

3.1 Classification algorithm

In this study, eleven (11) classification algorithms with diverse classification procedures are selected. Specifically, algorithms from Bayesian (NB and Bayesian Network (BN)), Instance-based learning (KNN and KStar (K*)), linear-based (SVM and LR), rule-based (Conjunctive Rule (CR) and Decision Table (DTab)) and tree-based (Random Tree (RT), DT and CART) classification family are selected and implemented for Android malware detection. The choice of these classifiers is primarily based on their respective performances and usage in existing ML studies [37]. Moreover, this study aims to extensively compare the performance of prominently deployed classifiers in Android malware detection. To the best of our knowledge, no existing study has investigated the effectiveness of the selected classifiers on Android malware datasets in the presence of a class imbalance problem.

Bayesian classification algorithms. Bayesian classification algorithms are primarily based on Bayesian Theorem. This set of classifiers can determine the probabilities of an instance belonging to a specific class. This is based on the notion that the Bayesian theorem depends on the assumption that given a predicted outcome, the features deployed for generating a prediction are independent of each other [38]. In this study, the duo of NB and BN are selected from the Bayesian classification algorithms. Both NB and BN have been extensively used in ML experiments across different research domains and are known to have good prediction performance [19, 39, 40]. Table 1 presents the parameters of NB and BN as deployed in this study.

Table 1. Parameter setting of selected Bayesian classification algorithms

Algorithms	Parameter Setting
Naïve Bayes (NB)	NumDecimalPlaces = 2; NumAttrEval = Normal Dist.
Bayesian Network (BN)	estimator = SimpleEstimator; searchAlgo = sim-pleBayes; useADTree = false

Instance-based learning classification algorithms. Instance-based learning classification algorithms are also known as memory-based or lazy classification algorithms. This set of algorithms outrightly compares new instances with already encountered instances stored in memory during model training [41]. That is, model evaluation is performed only when new instances are detected. In this study, KNN and K* algorithms are selected as instance-based learners. Both KNN and K* are instance-based learners, which means that the class of a test instance is decided by the class of training instances that are like it, as defined by some similarity function. In the case of K*, an entropy-based distance function is deployed as a similarity function while KNN is based on the Euclidean distance function [42, 43]. Table 2 shows the parameters of KNN and K* as deployed in this study.

Table 2. Parameter setting of selected instance-based learning classification algorithms

Algorithms	Parameter Setting
K Nearest Neighbour (KNN)	K=1; distanceWeighting= False; nearestNeigh bourSearch= LinearNNSearch
KStar (K*)	globalBlend=20; missingMode= AverageColumnEntropyCurves; entropicAutoBelnd=False

Linear-based classification algorithms. Linear-based classification algorithms are based on the value of a linear combination of the features. Linear-based classifiers in form of a predictor function combine a set of weights with these feature vectors. The SVM and LR are typical examples of linear-based classifiers [19]. Both algorithms (SVM and LR) are investigated in this study. In SVM, classification tasks are done by determining hyper-plane that distinguishes class labels. LR in its case fits a logistic function (sigmoid function) for the classification process. Table 3 shows a description of the parameters of SVM and LR as used in this study.

Table 3. Parameter setting of selected linear-based classification algorithms

Algorithms	Parameter Setting
Support Vector Machine (SVM)	kernelType=RBF; eps=0.001;loss=0.1; nu=0.5; shrinking= true; cost=1.0
Logistic Regression (LR)	Maxlts= -1;ridge= 1.0E-8; useConjugateGradientDescent=False

Rule-based classification algorithms. Rule-based classification algorithms make use of the if-then construct for predicting class labels. Due to the simplicity and ease of understanding of these rules, these classifiers are commonly deployed to develop descriptive models. In this study, the CR and DTab are deployed as instances of rule-based classifiers. The CR method employs a unique rule learner capable of predicting

class labels by utilizing the "AND" operator to connect data features (antecedents and consequents). DTab on the other hand is made up of two parts: a schema, which is a collection of attributes that are included in the table, and a body, which is made up of labelled instances from the space defined by the schema's attributes [44]. Table 4 presents the parameter setting of the selected rule-based learners.

Table 4. Parameter setting of selected rule-based classification algorithms

Algorithms	Parameter Setting
Conjunctive Rule (CR)	Exclusive=False; minNo=2.0; numAntds=-1;
Decision Table (DTab)	evaluationMeasure=Accuracy, RMSE; Search=BestFirst; useIBK=False;

Tree-based classification algorithms. Tree-based algorithms are widely regarded as one of the best and most often used ML algorithms. Due to their sophistication, three (3) tree-based classifiers are used in this study. Specifically, RT, DT as well as CART are investigated. RT employs a gripping concept to generate a collection of random data to construct a tree. Nearly every node in the parent tree is divided using the best split across all features. Each node in the RT is split using a best among the group of predictors randomly picked at that node. Similarly, DT is based on a deviation of information gain (IG), which is commonly used to evaluate the outcome of biasness. DT partitions data into subsets to map the training dataset into the smallest tree. As a splitting attribute, an attribute with the highest gain ratio is nominated in the direction of shaping a tree. The CART is a tree-based algorithm that is used to construct a decision tree based on Gini's impurity index as its splitting criterion. It is a simple machine learning method with a wide range of applications. CART as a binary tree is built on the iterative splitting of its node into two child nodes. Table 5 shows the parameters of RT, DT and CART as deployed in this study.

Table 5. Parameter setting of selected rule-based classification algorithms

Algorithms	Parameter Setting
Random Tree (RT)	KValue=0; minNum=1.0; minVarianceProp=0.001; maxDepth=0; breakTiesRandomly=False
Decision Tree (DT)	confidenceFactor=0.25; minNumObj=2; subTreeraising=True; useMDLcorrection= True
Classification and Regression Tree (CART)	Heuristic= True; minObjNum=2.0; numFoldsPruning=5; SizePer=1.0; usePrune=True; UseOneSE=False

3.2 Android malware datasets

During the experimental section of this research, two Android malware datasets were employed. These datasets are widely available and often utilized in current research [6, 32, 45–47]. The first dataset (Drebin) consists of 15,036 instances (5,560 malware and 9476 benign). Drebin has 215 independent features that describe the dataset. The second dataset (Malgenome) has 3.799 instances divided into 1,260 malware and 2.539 benign instances. Similar to Drebin, Malgenome has 215 features extracted from the

Android malware genome project [3]. For more information on the malware datasets, refer to [3, 48]. Table 6 displays features of the experimented (Drebin and Malgenome) datasets.

Table 6. Details of Android malware datasets

Algorithms	Number of Features	Number of Instances	Number of Benign	Number of Malware
Drebin	215	15,036	9,476	5,560
Malgenome	215	3.799	2,539	1,260

3.3 Performance evaluation metrics

In this study, Accuracy, F-measure, and Area under the Curve (AUC) are used to assess the detection performances of experimented Android malware models. The selection of these assessment measures is based on current research that shows widespread and regular use of these evaluation metrics for Android malware detection.

3.4 Experimental procedure

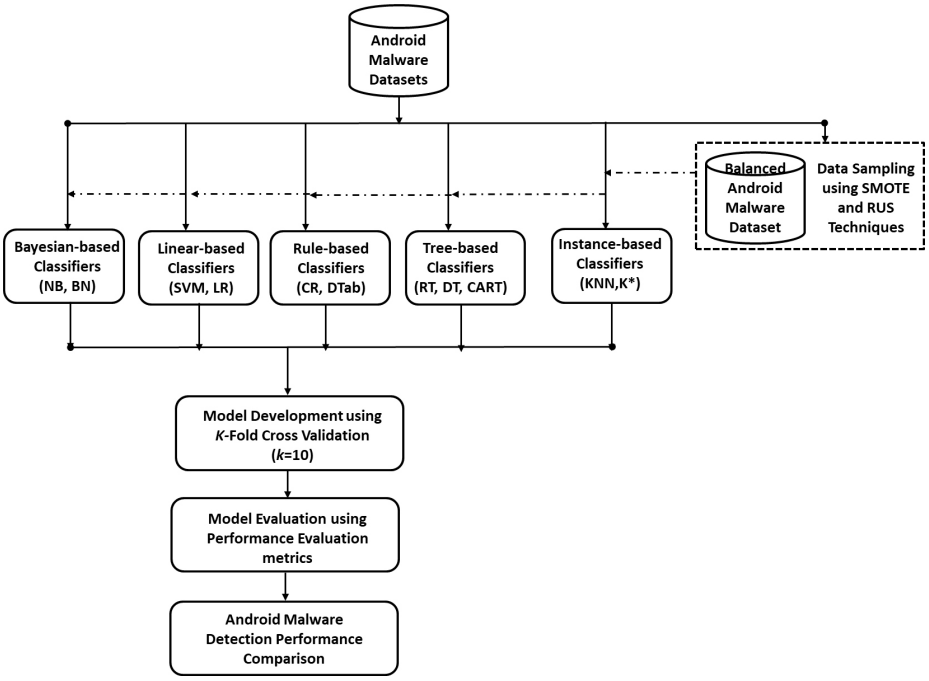

Fig. 1. Experimental procedure

This section discusses the experimental procedure presented in Figure 1 as followed in this research. The experimental procedure is structured to empirically test and verify the efficacy of the selected ML methods (See Section 3.1) for Android malware detection. Specifically, the selected ML methods are deployed on the original Android malware datasets. Findings from these experiments will empirically validate the performance of the selected ML methods for Android malware detection. Furthermore, to investigate the effect of class imbalance in Android malware detection, the inherent class imbalance in the Android malware datasets will be resolved using data sampling methods. In other words, a data over-sampling method (SMOTE) and under-sampling (RUS) methods are deployed used to balance the studied malware datasets. SMOTE and RUS are prominent data sampling methods that are used to address the class imbalance problem [19, 20, 49]. Thereafter, the selected ML methods are used on the balanced datasets. Experimental observations from these experiments will illustrate the effect of data sampling methods on ML methods in Android malware detection. For training and evaluating selected ML methods, the K-fold (where k = 10) cross-validation (CV) approach is used for the creation and evaluation of the malware detection models. The 10-fold CV option is based on its ability to create malware detection models with the low impact of the issue of class imbalance [20, 50–53]. Moreover, the K-fold CV approach ensures that each instance can be used iteratively for both training and testing [54–56]. The Waikato Environment for Knowledge Analysis (WEKA) machine learning library [57] and R programming language [58] are used for the experimentation on an Intel(R) Core™ machine equipped with i7-6700, running at speed 3.4 GHz CPU with 16 GB RAM.

4 Experimental results

This section presents the analysis of experimental results of the eleven (11) ML methods selected from five families (Bayesian, Linear-based, Rule-based, Tree-based and Instance-based methods) for Android malware detection. A total of 66 experimental scenarios with each experiment repeated 10 times (660 experiments) were conducted to derive the results. The experiments were performed in the order as explained in Section 3.4. Initially, the selected ML methods were applied on the original Malgenome and Drebin datasets to evaluate their respective detection performances in the presence of a class imbalance problem. Subsequently, the inherent class imbalance present in the original datasets were resolved using SMOTE and RUS methods. That is, instances (minority and majority) present in both Malgenome and Drebin are balanced using SMOTE and RUS methods. Consequently, each of the selected ML methods was deployed on the newly generated (balanced) Android malware datasets. The essence of this procedure is to empirically ascertain the suitability of sampling methods in Android malware detection. Table 7 and Table 8 presents the results of the experimented ML methods on the original Malgenome and Drebin datasets respectively.

Table 7. The detection performance of experimented ML methods on the Malgenome dataset

	Accuracy	AUC	F-Measure
NB	92.58	0.926	0.926
BN	92.73	0.927	0.927
SVM	97.81	0.969	0.969
LR	96.87	0.969	0.969
CR	75.44	0.754	0.754
DTab	91.81	0.911	0.911
RT	96.24	0.950	0.950
DT	97.21	0.941	0.941
CART	97.21	0.967	0.967
KNN	98.24	0.955	0.955
K*	97.16	0.944	0.944

Table 8. The detection performance of experimented ML methods on the Drebin dataset

	Accuracy	AUC	F-Measure
NB	82.42	0.824	0.824
BN	82.78	0.828	0.828
SVM	96.00	0.964	0.964
LR	96.81	0.978	0.978
CR	75.55	0.756	0.758
DTab	90.20	0.922	0.922
RT	95.30	0.952	0.952
DT	95.33	0.975	0.973
CART	95.28	0.973	0.973
KNN	96.76	0.988	0.988
K*	96.80	0.988	0.988

As shown in Table 7, it can be observed that the duo of KNN and SVM had the best performance on the Malgenome dataset. In particular, KNN and SVM had detection accuracy values of 98.24% and 97.81% respectively. Between the Bayesian-based classifiers, both BN and NB performed comparably with detection accuracy values of 92.78% and 92.58% respectively. In the case of linear-based classifiers, SVM and LR recorded 97.81% and 96.87% accuracy values accordingly. CR and DTab, which are rule-based classifiers performed relatively well. Specifically, DTab (91.81%) had a better detection accuracy as compared with CR (75.44%). The poor performance of CR can be linked to the high number of cycles needed to attain minimum requirements. Also, the tree-based classifiers (RT, CART, and DT) performed well with a detection accuracy value of 97.21% (DT), 97.21% (CART), and 96.24% (RT). On average, the instance-based methods (KNN and K*) recorded the best performance on the Malgenome dataset compared to ML methods. This feat can be due to their ability to

withstand redundant and noisy features [41, 42]. Table 8 presents the detection performances of experimented ML methods on the Drebin dataset. Similar to observations on the Malgenome dataset, the instance-based classifiers had the best Android malware detection performance on the Drebin dataset. The duo of KNN and K* classifiers had detection accuracy values of 96.76% and 96.80% respectively. Also, the linear-based classifiers (SVM and LR) performed comparably to the instance-based classifiers (KNN and K*). The tree-based classifiers equal performed well with a minimum detection accuracy of 95.28% by CART. NB and BN as Bayesian-based classifiers performed relatively well as compared with instance and linear-based classifiers. However, the performance of the rule-based classifiers is somewhat indifferent. CR still maintains its poor detection performance with 75.55% Android malware detection accuracy while DTab had a 90.20% accuracy value.

The respective performances of the experimented ML methods on the Malgenome and Drebin datasets showed positive outcomes. Specifically, aside from CR classifiers, all experimented ML methods had detection accuracy and AUC values greater than 80% and 90% respectively. This observation indicates that the experimented ML methods can perform well in Android malware detection. However, knowing the adverse effect of the cost of misclassification or misdetection of Android malware, it is pertinent to develop a highly sophisticated ML model. Reported findings from existing ML studies have shown that the quality of the dataset used in an ML task has a direct effect on the performance of selected ML methods. As such, the class imbalance problem has been established as an inherent data quality problem present in Android malware datasets. Based on this premise, data sampling methods (SMOTE and RUS) are used to resolve the inherent imbalance in studied Android malware datasets. Table 9 and Table 10 shows the performance of experimented ML methods on balanced (SMOTE) Malgenome and Drebin datasets respectively. The essence of these analyses is to investigate the effect of the studied data sampling methods (SMOTE and RUS) on ML methods in Android malware detection.

Table 9. The detection performance of experimented ML methods on balanced (SMOTE) Malgenome dataset

	Accuracy	**AUC**	**F-Measure**
NB	95.26	0.994	0.953
BN	94.50	0.994	0.945
SVM	98.24	0.982	0.982
LR	99.33	0.998	0.993
CR	81.42	0.809	0.809
DTab	93.64	0.986	0.936
RT	97.90	0.979	0.979
DT	98.06	0.983	0.981
CART	98.63	0.980	0.976
KNN	99.49	0.998	0.995
K*	98.83	0.997	0.988

Table 10. The detection performance of experimented ML methods on balanced (SMOTE) Drebin dataset

	Accuracy	**AUC**	**F-Measure**
NB	85.61	0.945	0.945
BN	85.58	0.942	0.942
SVM	97.24	0.972	0.972
LR	98.1	0.996	0.981
CR	80.1	0.84	0.84
DTab	92.41	0.976	0.976
RT	97.74	0.978	0.977
DT	97.77	0.982	0.982
CART	97.65	0.984	0.984
KNN	99.09	0.997	0.997
K*	98.61	0.998	0.998

As with the Malganome dataset, it can be observed from Table 9 that considering the performance within ML methods, LR, DTab, and KNN outperformed other experimented ML methods in Accuracy, AUC, and F-measure values. NB performed better than BN in accuracy and F-measure while both performed the same on AUC values. Concerning tree-based classifiers, DT outperformed CART and RT in AUC and F-measure values, although CART outperformed both RT and DT on detection accuracy values. Overall, KNN had the best performance while CR had the least performance. On the Drebin dataset balanced with SMOTE data over-sampling method, Table 10 shows that NB, LR, and DTab outperformed other experimented ML methods. KNN did better than K* in terms of Accuracy, although K* had slightly better AUC and F-measure values. Like the balanced Malgenome dataset, KNN recorded the best performance while CR had the least Android malware detection performance.

Furthermore, concerning the enhancement in detection performance of the experimented ML methods with the application of SMOTE data balancing method, Figure 2 and Figure 3 presented the variation (percentage increase) in Android malware detection performance measures recorded on Malganome and Debrin datasets respectively.

With the Malgenome dataset, generally, there are considerable improvements in the performance of approximately all the experimented ML methods in all performance measures after balancing with SMOTE oversampling method. In terms of Accuracy, CR which was the worst-performing ML method in the presence of class imbalance recorded the highest percentage increase achieving approximately +8% improvement in accuracy value. This shows the extent to which data balancing can enhance the performance of poor performing models by alleviating class imbalance present in the Android malware dataset. NB and LR recorded approximately a +3% increase in detection accuracy while other experimented ML methods were able to achieve less than +2% increase in malware detection accuracy values.

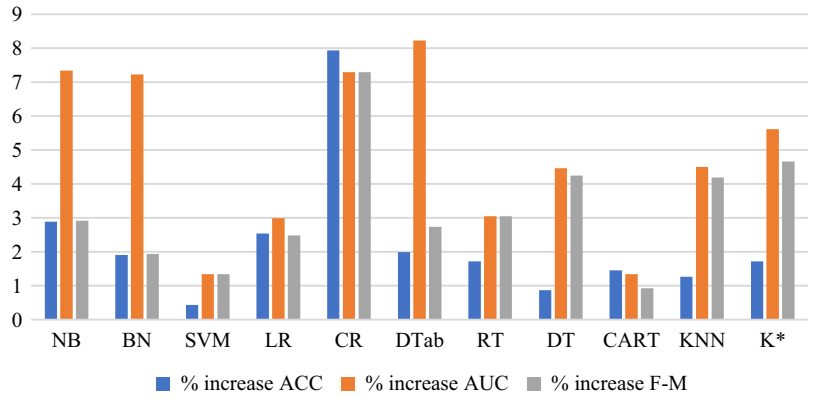

Fig. 2. Malware detection performance variation of experimented ML method on SMOTE-balanced Malgenome dataset

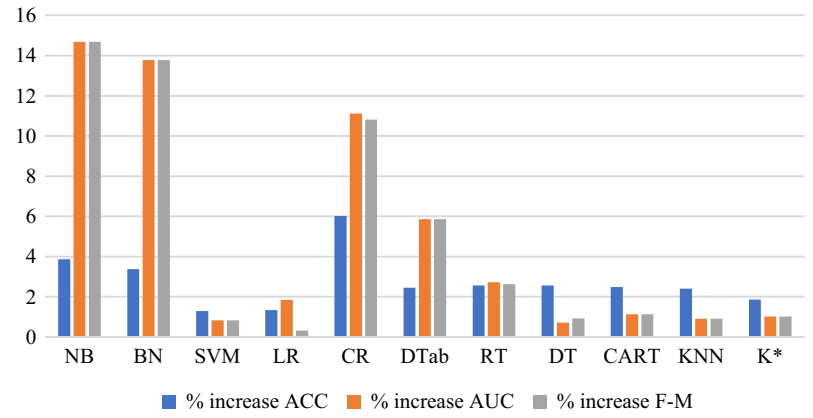

Fig. 3. Malware detection performance variation of experimented ML method on SMOTE-balanced Drebin dataset

Specifically, the effect of SMOTE balancing method is comparably negligible on the performance of SVM as it achieved only a +0.44% increase in detection accuracy value. In terms of AUC, DTab produced the highest performance improvement recording more than +8% increment and followed slightly by CR, NB and BN with a recorded increase greater than 7%. DT, KNN, and K* achieved more than 4% increment while LR and RT achieved a +3% increase. SVM and CART showed very little improvement in AUC value as both recorded a +1.34% increase. In terms of F-Measure, CR also recorded the highest percentage increase in performance achieving a +7.3% increase. Following CR in improvement are K*, KNN and DT with more than +4% increase in F-Measure. RT,

DTab, NB and LR achieved more than a +2% increase in F-Measure. CART and SVM recorded the smallest percentage increase in F-Measure recording as low as +0.93% and +1.24% increase respectively. It is also observed that the rule-based approaches improved the most across all performance measures followed by the Bayesian-based and instance-based approaches. Linear-based and Tree-based classifiers showed a little improvement except for DT which achieved more than +4% increase in AUC and F-Measure values.

On the Drebin dataset, SMOTE oversampling method also showed a positive enhancement in the detection performance of experimented ML methods. In terms of AUC and F-Measure values, NB achieved the highest performance increase recording over +14% increment in both measures. BN followed closely in percentage increase in AUC and F-Measure with over +13% increase in both measures. Similarly, CR and DTab showed a significant improvement in AUC and F-measure values. In terms of Accuracy, CR produced a better increase in performance than other ML methods achieving a +6% increment. NB and BN also achieved a good performance increment in accuracy value (more than +3% increment). As observed with the Malgenome dataset, SVM had the least increment in performance based on detection accuracy and AUC values while LR had the least performance based on F-measure. Among the experimented ML methods, the Bayesian-based classifiers had the highest increment in detection performance followed by the rule-based classifiers.

Table 11. The detection performance of experimented ML methods on balanced (RUS) Malgenome dataset

	Accuracy	**AUC**	**F-Measure**
NB	95.52	0.994	0.955
BN	94.33	0.993	0.943
SVM	97.74	0.977	0.977
LR	98.77	0.997	0.988
CR	80.83	0.808	0.804
DTab	94.64	0.98	0.946
RT	96.83	0.968	0.968
DT	96.79	0.974	0.968
CART	98.71	0.972	0.967
KNN	99.13	0.996	0.991
K*	99.09	0.999	0.991

Table 12. The detection performance of experimented ML methods on balanced (RUS) Drebin dataset

	Accuracy	AUC	F-Measure
NB	85.66	0.95	0.856
BN	85.45	0.929	0.853
SVM	96.93	0.969	0.969
LR	97.59	0.994	0.976
CR	80.06	0.803	0.795
DTab	93.71	0.98	0.937
RT	96.56	0.966	0.966
DT	96.99	0.976	0.97
CART	96.64	0.978	0.966
KNN	98.38	0.994	0.984
K*	98.48	0.998	0.985

Similar to the SMOTE method, the RUS method was also deployed to balance the Malgenome and Drebin datasets. Table 11 and Table 12 show the results of experimented ML methods on the balanced (RUS) Android malware datasets. From Table 11, the experimental results showed that NB, LR, DTab had the best malware detection performance on the newly generated RUS-based Malgenome dataset. Specifically, Among the tree-based classifiers, CART had the highest malware detection accuracy value, DT recorded the most superior AUC and F-measure values. Also, Table 12 showed the detection performance of experimented ML methods on the RUS-balanced Drebin dataset. NB, LR, DTab, K*, and DT performed better than other ML methods. K* and CR had the overall best and worst Android malware detection performance respectively among the experimented ML methods.

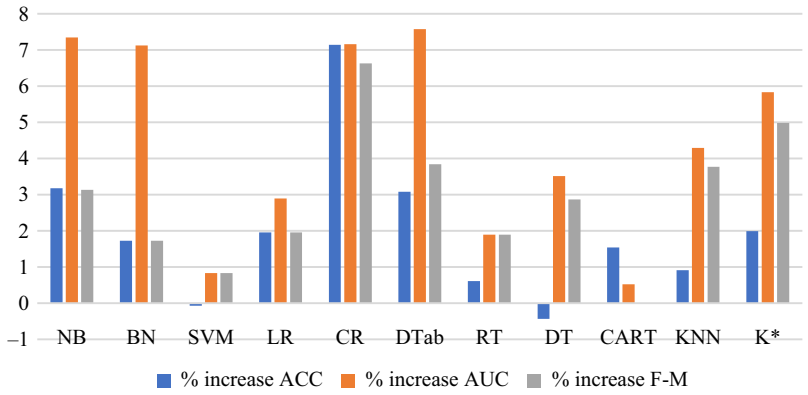

Fig. 4. Malware detection performance variation of experimented ML method on RUS-balanced Malgenome dataset

Also, concerning the effect of the RUS method on the Android malware detection performances of the experimented ML methods, Figure 4 and Figure 5 present the variation in detection performance measures recorded on the Malgenome and Drebin datasets respectively.

As presented in Figure 4, due to the alleviated class imbalance problem by deploying the RUS method on the Malgenome dataset, there were enhancements (positive increment) in performance measures of the experimented ML methods. The positive increment in performance metric values occurred for the experimented ML methods except in the case of SVM and DT classifiers whose detection accuracy values decreased by 0.07% and 0.43% respectively.

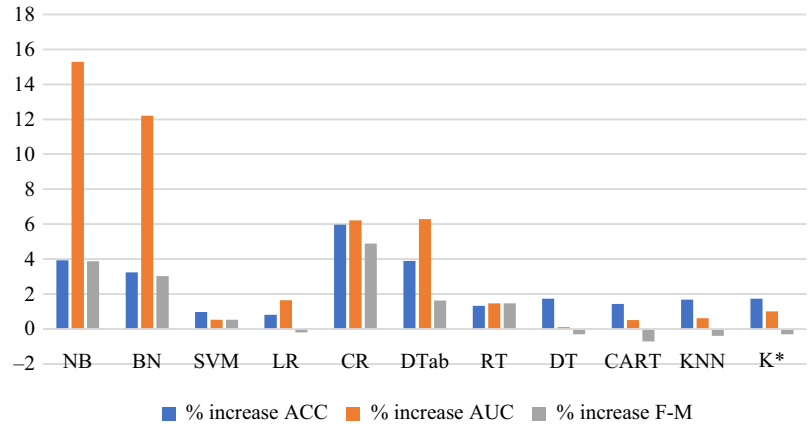

Fig. 5. Malware detection performance variation of experimented ML method on RUS-balanced Drebin dataset

Specifically, the CR classifier had the highest positive variation in detection accuracy and F-Measure values achieving +7.14% and +6.63% increment respectively. NB and KNN classifiers recorded high positive performance variations in detection accuracy and f-measure values respectively. In terms of AUC value, DTab recorded the highest percentage increase (+7.57%) followed closely by NB (+7.34%), CR (+7.16%) and BN (+7.12%) classifiers respectively. CART and SVM had the least variations (less than +1%) in AUC values. Similar to the findings observed with the SMOTE-based Malgenome dataset, it can be observed that the detection performances of experimented ML methods improved significantly with the deployment of the RUS method. The rule-based classifiers improved the most in Android malware detection performance followed by Bayesian-based classifiers and instance-based classifiers. The least improvement in Android malware detection performance is observed in the linear-based and tree-based classifiers. In the case of the RUS method on the Drebin dataset, Figure 5 illustrates the Android malware detection performance variation of experimented ML method on the RUS-balanced Drebin dataset. Positive variations (increment) were observed in detection accuracy and AUC values of all experimented ML methods. Although, there was a decrease in the F-measure values of some experimented ML methods (LR, DT, CRT, KNN and K*), this can be attributed to the information loss often caused by the

RUS method. More specifically, NB (+15.29%) and BN (+12.20%) classifiers achieved the highest percentage increase in AUC values respectively while the CR (+5.97%) classifier achieved the highest percentage increment in detection accuracy value. The F-Measure result is poor with the RUS-based Drebin dataset, nevertheless, the CR classifier still recorded the highest percentage achieving approximately a +5% increment.

Lastly, the overall effects of deploying an oversampling method (SMOTE) and undersampling method (RUS) on the performance of the experimented ML methods was observed. The AUC metric is selected for the evaluation as it is considered the most informative measure for classification tasks. Figure 6 and Figure 7 presents the experimental results comparison of ML models based on SMOTE and RUS-based Malgenome and Drebin Android malware datasets respectively.

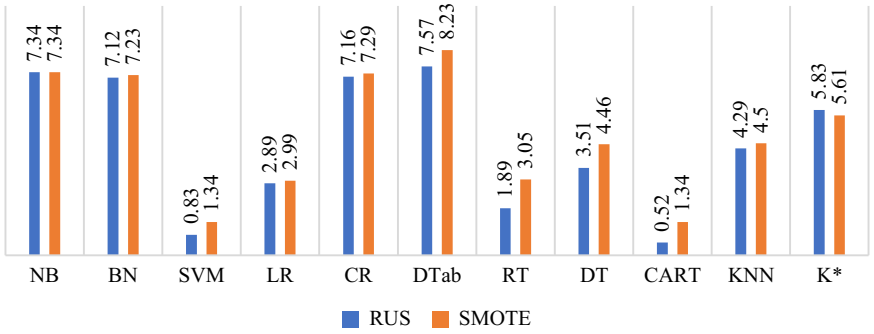

Fig. 6. ML methods malware detection performance enhancement comparison of RUS and SMOTE-based Malgenome dataset

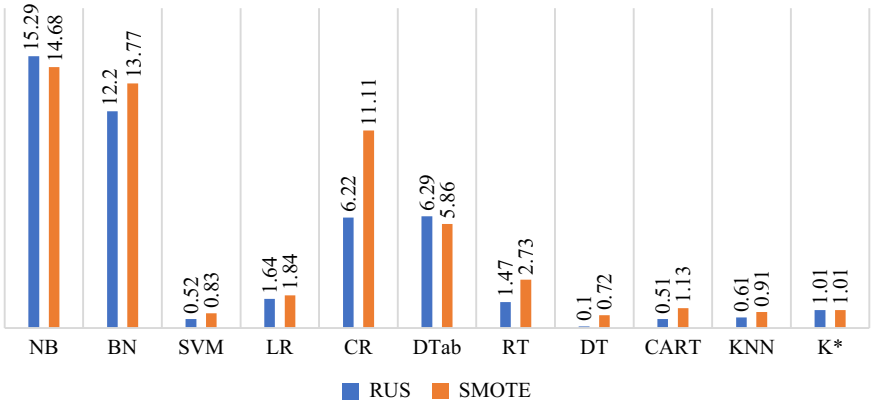

Fig. 7. ML methods malware detection performance enhancement comparison of RUS and SMOTE-based on Drebin dataset

From Figure 6, it can be observed that SMOTE had superior positive effects on the detection performances of the experimented ML methods than RUS except in NB where they have comparable effects. This superior performance is most significant in

CR classifier. A similar finding can be observed from Figure 7 as SMOTE performs better than RUS in all cases except in NB where RUS supersedes SMOTE. Hence, comparatively, it can be concluded that the oversampling method (SMOTE) is superior to the under-sampling method (RUS) for addressing the class imbalance problem and enhancing the detection performance of ML methods in android malware detection.

5 Conclusions and future works

This work conducted an extensive comparative analysis on the performance of eleven (11) ML methods - selected from five different families with diverse and distinct computational characteristics - in the presence of the class imbalance problem for Android malware detection. Specifically, NB, BN, SVM, LR, CR, DTab, DT, RT, CART, KNN and K* classification algorithms are deployed on original (imbalanced) and balanced Android malware datasets. The SMOTE and RUS methods were used as data sampling methods to resolve (balance) the class imbalance nature of the Android malware datasets. Drebin and Malgenome Android malware datasets were employed to test the performance of the experimented ML methods. Findings from the empirical results and analyses indicated that the instance-based methods (KNN and K*) have inculpable resistance to the class imbalance problem that may exist in Android malware datasets. Also, it was observed that studied data sampling methods (SMOTE and RUS) positively enhanced the detection performances of the experimented ML methods. Lastly, this study confirmed the superiority of SMOTE method over the RUS method in addressing the class imbalance problem and improving the detection performance of ML methods in Android malware detection tasks. As a limitation for this study, more real-life datasets would be considered in future works. Also, the high dimensionality of Android malware datasets is another issue that needs to be addressed. The combination of data sampling and feature selection methods as solutions to class imbalance and high dimensionality problems in Android malware detection will be investigated.

6 References

[1] M. E. Khoda, J. Kamruzzaman, I. Gondal, T. Imam, and A. Rahman, "Malware detection in edge devices with fuzzy oversampling and dynamic class weighting," *Applied Soft Computing*, vol. 112, p. 107783, 2021. https://doi.org/10.1016/j.asoc.2021.107783

[2] A. Alsarhan, A.-R. Al-Ghuwairi, E. Alshdaifat, and H. Idhaim, "A novel scheme for malicious nodes detection in cloud markets based on fuzzy logic technique," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 3, 2022. https://doi.org/10.3991/ijim.v16i03.27933

[3] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018. https://doi.org/10.1109/TCYB.2017.2777960

[4] A. Abozeid, A. A. AlHabshy, and K. ElDahshan, "A Software Security Optimization Architecture (SoSOA) and its adaptation for mobile applications," *International Journal of Interactive Mobile Technologies*, vol. 15, no. 11, 2021. https://doi.org/10.3991/ijim.v15i11.20133

[5] F. Alswaina and K. Elleithy, "Android malware family classification and analysis: Current status and future directions," *Electronics*, vol. 9, no. 6, p. 942, 2020. https://doi.org/10.3390/electronics9060942

[6] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48–S59, 2018. https://doi.org/10.1016/j.diin.2018.01.007

[7] A. G. Oladepo, A. O. Bajeh, A. O. Balogun, H. A. Mojeed, A. A. Salman, and A. I. Bako, "Heterogeneous ensemble with combined dimensionality reduction for social spam detection," *International Journal of Interactive Mobile Technologies*, vol. 15, no. 17, 2021. https://doi.org/10.3991/ijim.v15i17.19915

[8] Kaspersky. (2021, 21/12/2021). *Mobile malware evolution 2020*. Available: https://securelist.com/mobile-malware-evolution-2020/101029/

[9] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A review of android malware detection approaches based on machine learning," *IEEE Access*, vol. 8, pp. 124579–124607, 2020. https://doi.org/10.1109/ACCESS.2020.3006143

[10] I. Almomani *et al.*, "Android ransomware detection based on a hybrid evolutionary approach in the context of highly imbalanced data," *IEEE Access*, vol. 9, pp. 57674–57691, 2021. https://doi.org/10.1109/ACCESS.2021.3071450

[11] R. Almohaini, I. Almomani, and A. AlKhayer, "Hybrid-based analysis impact on ransomware detection for Android systems," *Applied Sciences,* vol. 11, no. 22, p. 10976, 2021. https://doi.org/10.3390/app112210976

[12] M. Dhalaria and E. Gandotra, "Android malware detection using chi-square feature selection and ensemble learning method," in *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, 2020, pp. 36–41: IEEE. https://doi.org/10.1109/PDGC50313.2020.9315818

[13] M. Dhalaria and E. Gandotra, "A framework for detection of Android malware using static features," in *2020 IEEE 17th India Council International Conference (INDICON)*, 2020, pp. 1–7: IEEE. https://doi.org/10.1109/INDICON49873.2020.9342511

[14] P. Agrawal and B. Trivedi, "Machine learning classifiers for Android malware detection," in *Data Management, Analytics and Innovation*: Springer, 2021, pp. 311–322. https://doi.org/10.1007/978-981-15-5616-6_22

[15] A. Amouri, V. T. Alaparthy, and S. D. Morgera, "A machine learning based intrusion detection system for mobile internet of things," *Sensors*, vol. 20, no. 2, p. 461, 2020. https://doi.org/10.3390/s20020461

[16] M. S. Hussain and K. U. R. Khan, "A survey of ids techniques in manets using machine-learning," in *Third International Conference on Computational Intelligence and Informatics. Springer*, 2020, pp. 743–751. https://doi.org/10.1007/978-981-15-1480-7_68

[17] Y. A. Alsariera, V. E. Adeyemo, A. O. Balogun, and A. K. Alazzawi, "Ai meta-learners and extra-trees algorithm for the detection of phishing websites," *IEEE Access*, vol. 8, pp. 142532–142542, 2020. https://doi.org/10.1109/ACCESS.2020.3013699

[18] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, "Performance analysis of feature selection methods in software defect prediction: a search method approach," *Applied Sciences*, vol. 9, no. 13, p. 2764, 2019. https://doi.org/10.3390/app9132764

[19] A. O. Balogun, S. Basri, S. J. Abdulkadir, V. E. Adeyemo, A. A. Imam, and A. O. Bajeh, "Software defect prediction: Analysis of class imbalance and performance stability," *Journal of Engineering Science and Technology,* vol. 14, no. 6, pp. 3294–3308, 2019.

[20] A. O. Balogun *et al.*, "SMOTE-based homogeneous ensemble methods for software defect prediction," in *International Conference on Computational Science and Its Applications*, 2020, pp. 615–631: Springer. https://doi.org/10.1007/978-3-030-58817-5_45

[21] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data,* vol. 5, no. 1, pp. 1–30, 2018. https://doi.org/10.1186/s40537-018-0151-6

[22] S. Sen, A. I. Aysan, and J. A. Clark, "SAFEDroid: using structural features for detecting android malwares," in *International Conference on Security and Privacy in Communication Systems*, 2017, pp. 255–270: Springer. https://doi.org/10.1007/978-3-319-78816-6_18

[23] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2014. https://doi.org/10.1109/TDSC.2014.2355839

[24] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications," in *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*, 2016, pp. 1–5: IEEE. https://doi.org/10.1109/NTMS.2016.7792435

[25] H. Rathore, S. K. Sahay, P. Chaturvedi, and M. Sewak, "Android malicious application classification using clustering," in *International Conference on Intelligent Systems Design and Applications*, 2018, pp. 659–667: Springer. https://doi.org/10.1007/978-3-030-16660-1_64

[26] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *2012 European Intelligence and Security Informatics Conference*, 2012, pp. 141–147: IEEE. https://doi.org/10.1109/EISIC.2012.34

[27] X. Su, M. Chuah, and G. Tan, "Smartphone dual defense protection framework: Detecting malicious applications in android markets," in *2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, 2012, pp. 153–160: IEEE. https://doi.org/10.1109/MSN.2012.43

[28] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using Bayesian classification," in *2013 IEEE 27th international conference on advanced information networking and applications (AINA)*, 2013, pp. 121–128: IEEE. https://doi.org/10.1109/AINA.2013.88

[29] Y. A. Alsariera, A. O. Balogun, V. E. Adeyemo, O. H. Tarawneh, and H. A. Mojeed, "Intelligent tree-based ensemble approaches for phishing website detection," *Journal of Engineering Science and Technology,* vol. 17, no. 1, pp. 0563–0582, 2022.

[30] S. S. M. M. Rahman and S. K. Saha, "StackDroid: Evaluation of a multi-level approach for detecting the malware on android using stacked generalization," in *International Conference on Recent Trends in Image Processing and Pattern Recognition*, 2018, pp. 611–623: Springer. https://doi.org/10.1007/978-981-13-9181-1_53

[31] M. Dhalaria and E. Gandotra, "CSForest: An approach for imbalanced family classification of android malicious applications," *International Journal of Information Technology*, vol. 13, no. 3, pp. 1059–1071, 2021. https://doi.org/10.1007/s41870-021-00661-7

[32] M. S. Rana and A. H. Sung, "Malware analysis on Android using supervised machine learning techniques," *International Journal of Computer and Communication Engineering*, vol. 7, no. 4, p. 178, 2018. https://doi.org/10.17706/IJCCE.2018.7.4.178–188

[33] S. A. Salihu, S. Quadri, and O. C. Abikoye, "Performance evaluation of selected machine learning techniques for malware detection in Android devices," *Ilorin Journal of Computer Science and Information Technology*, vol. 3, no. 1, pp. 52–61, 2020.

[34] L. K. Shar, B. F. Demissie, M. Ceccato, and W. Minn, "Experimental comparison of features and classifiers for android malware detection," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, 2020, pp. 50–60. https://doi.org/10.1145/3387905.3388596

[35] B. A. Gyunka, O. C. Abikoye, and A. S. Adekunle, "Anomaly Android malware detection: A comparative analysis of six classifiers," in *International Conference on Information and Communication Technology and Applications*, 2020, pp. 145–157: Springer. https://doi.org/10.1007/978-3-030-69143-1_12

[36] P. Agrawal and B. Trivedi, "Evaluating machine learning classifiers to detect android malware," in *2020 IEEE International Conference for Innovation in Technology (INOCON)*, 2020, pp. 1–6: IEEE. https://doi.org/10.1109/INOCON50539.2020.9298290

[37] A. O. Balogun *et al.*, "Optimized decision forest for website phishing detection," in *Proceedings of the Computational Methods in Systems and Software*, 2021, pp. 568–582: Springer. https://doi.org/10.1007/978-3-030-90321-3_47

[38] A. Tsymbal, S. Puuronen, and D. W. Patterson, "Ensemble feature selection with the simple Bayesian classification," *Information fusion*, vol. 4, no. 2, pp. 87–100, 2003. https://doi.org/10.1016/S1566-2535(03)00004-6

[39] G. D'Angelo, S. Rampone, and F. Palmieri, "Developing a trust model for pervasive computing based on Apriori association rules learning and Bayesian classification," *Soft Computing,* vol. 21, no. 21, pp. 6297–6315, 2017. https://doi.org/10.1007/s00500-016-2183-1

[40] A. O. Balogun *et al.*, "A novel rank aggregation-based hybrid multifilter wrapper feature selection method in software defect prediction," *Computational Intelligence and Neuroscience*, vol. 2021, 2021. https://doi.org/10.1155/2021/5069016

[41] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991. https://doi.org/10.1007/BF00153759

[42] M. A. Mabayoje, A. O. Balogun, H. A. Jibril, J. O. Atoyebi, H. A. Mojeed, and V. E. Adeyemo, "Parameter tuning in KNN for software defect prediction: An empirical analysis," *Jurnal Teknologi dan Sistem Komputer,* vol. 7, no. 4, pp. 121–126, 2019. https://doi.org/10.14710/jtsiskom.7.4.2019.121-126

[43] D. Y. Mahmood and M. A. Hussein, "Intrusion detection system based on K-star classifier and feature set reduction," *International Organization of Scientific Research Journal of Computer Engineering (IOSR-JCE) Vol,* vol. 15, no. 5, pp. 107–112, 2013. https://doi.org/10.9790/0661-155107112

[44] A. O. Balogun *et al.*, "Rank aggregation based multi-filter feature selection method for software defect prediction," in *International Conference on Advances in Cyber Security*, 2020, pp. 371–383: Springer. https://doi.org/10.1007/978-981-33-6835-4_25

[45] M. S. Rana, C. Gudla, and A. H. Sung, "Evaluating machine learning models for Android malware detection: A comparison study," in *Proceedings of the 2018 VII International Conference on Network, Communication and Computing*, 2018, pp. 17–21. https://doi.org/10.1145/3301326.3301390

[46] M. S. Rana, S. S. M. M. Rahman, and A. H. Sung, "Evaluation of tree based machine learning classifiers for android malware detection," in *International Conference on Computational Collective Intelligence*, 2018, pp. 377–385: Springer. https://doi.org/10.1007/978-3-319-98446-9_35

[47] M. S. Rana and A. H. Sung, "Evaluation of advanced ensemble learning techniques for Android malware detection," *Vietnam Journal of Computer Science*, vol. 7, no. 02, pp. 145–159, 2020. https://doi.org/10.1142/S2196888820500086

[48] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in *Ndss*, 2014, vol. 14, pp. 23–26. https://doi.org/10.14722/ndss.2014.23247

[49] A. O. Balogun, K. S. Adewole, A. O. Bajeh, and R. G. Jimoh, "Cascade generalization based functional tree for website phishing detection," in *International Conference on Advances in Cyber Security*, 2021, pp. 288–306: Springer. https://doi.org/10.1007/978-981-16-8059-5_17

[50] A. O. Balogun, A. O. Bajeh, V. A. Orie, and W. A. Yusuf-Asaju, "Software defect prediction using ensemble learning: An ANP based evaluation method," *FUOYE Journal of Engineering and Technology,* vol. 3, no. 2, pp. 50–55, 2018. https://doi.org/10.46792/fuoyejet.v3i2.200

[51] R. Jimoh, A. Balogun, A. Bajeh, and S. Ajayi, "A PROMETHEE based evaluation of software defect predictors," *Journal of Computer Science and Its Application*, vol. 25, no. 1, pp. 106–119, 2018.
[52] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," presented at the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), 2016. https://doi.org/10.1109/ISSRE.2016.13
[53] Q. Yu, S. Jiang, and Y. Zhang, "The performance stability of defect prediction models with class imbalance: An empirical study," *IEICE TRANSACTIONS on Information and Systems*, vol. 100, no. 2, pp. 265–272, 2017. https://doi.org/10.1587/transinf.2016EDP7204
[54] S. Yadav and S. Shukla, "Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification," presented at the 2016 IEEE 6th International conference on advanced computing (IACC), 2016. https://doi.org/10.1109/IACC.2016.25
[55] S. Arlot and M. Lerasle, "Choice of V for V-fold cross-validation in least-squares density estimation," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7256–7305, 2016.
[56] A. O. Balogun *et al.*, "Search-based wrapper feature selection methods in software defect prediction: An empirical analysis," in *Computer Science On-line Conference*, 2020, pp. 492–503: Springer. https://doi.org/10.1007/978-3-030-51965-0_43
[57] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD explorations newsletter,* vol. 11, no. 1, pp. 10–18, 2009. https://doi.org/10.1145/1656274.1656278
[58] M. J. Crawley, *The R book*. John Wiley & Sons, 2012. https://doi.org/10.1002/9781118448908

7 Authors

Abimbola Ganiyat Akintola is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. She received M.Sc. and Ph.D. in Computer Science from University of Ilorin, Ilorin, Nigeria. She can be reached via her email address (akintola.ag@unilorin.edu.ng).

Abdullateef Oluwagbemiga Balogun received his B.Sc. and M.Sc degrees in Computer Science from University of Ilorin, Nigeria. He had his Ph.D. in Information Technology at the Universiti Teknologi PETRONAS, Perak, Malaysia. His research interests include Search-Based Software Engineering, Software Quality Assurance, Machine Learning, Data Science. He can be reached via his email address (balogun.ao1@unilorin.edu.ng; abdullateef_16005851@utp.edu.my).

Hammed Adeleye Mojeed is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. He received Master of Science and Bachelor of Science in Computer science from University of Ilorin, Ilorin, Nigeria. He is currently doing his PhD studies at the Institute of Technical Informatics and Telecommunication (ITiT), Gdansk University of Technology (GUT), Poland. His research interests fall in the field of Empirical Search-Based Software Engineering, Software Project Planning and Management and Machine Learning. He can be reached via his email address (mojeed.ha@unilorin.edu.ng).

Fatima Enehezei Usman-Hamza is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. She received M.Sc. and Ph.D. in Computer Science from University of Ilorin, Ilorin, Nigeria. She can be reached via her email address (usman-hamza.fe@unilorin.edu.ng).

Shakirat Aderonke Salihu is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. She received M.Sc. and Ph.D. in Computer Science from University of Ilorin, Ilorin, Nigeria. She can be reached via her email address (salihu.sa@unilorin.edu.ng).

Kayode Sakariyau Adewole is a Senior Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. He can be reached via his email address (adewole.ks@unilorin.edu.ng).

Ghaniyyat Bolanle Balogun is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. She received M.Sc. and Ph.D. in Computer Science from University of Ilorin, Ilorin, Nigeria. She can be reached via her email address (balogun.gb@unilorin.edu.ng).

Peter Ogirima Sadiku is a Lecturer in the Department of Computer Science, University of Ilorin, Ilorin Nigeria. He received M.Sc. in Computer Science from University of Ilorin, Ilorin, Nigeria. He can be reached via his email address (sadiku.po@unilorin.edu.ng).

Article submitted 2022-01-23. Resubmitted 2022-03-12. Final acceptance 2022-04-12. Final version published as submitted by the authors.