applied sciences

MDPI

Article

Empirical Analysis of Forest Penalizing Attribute and Its Enhanced Variations for Android Malware Detection

Abimbola G. Akintola [1], Abdullateef O. Balogun [1,2,*], Luiz Fernando Capretz [3], Hammed A. Mojeed [1,4], Shuib Basri [2], Shakirat A. Salihu [1], Fatima E. Usman-Hamza [1], Peter O. Sadiku [1], Ghaniyyat B. Balogun [1] and Zubair O. Alanamu [1]

[1] Department of Computer Science, University of Ilorin, Ilorin 1515, Nigeria; akintola.ag@unilorin.edu.ng (A.G.A.); mojeed.ha@unilorin.edu.ng (H.A.M.); salihu.sa@unilorin.edu.ng (S.A.S.); usman-hamza.fe@unilorin.edu.ng (F.E.U.-H.); sadiku.po@unilorin.edu.ng (P.O.S.); balogun.gb@unilorin.edu.ng (G.B.B.); alanamu.zo@unilorin.edu.ng (Z.O.A.)
[2] Department of Computer and Information Science, Universiti Teknologi PETRONAS, Bandar Seri Iskandar 32610, Perak, Malaysia; shuib_basri@utp.edu.my
[3] Department of Electrical and Computer Engineering, Western University, London, ON N6A 5B9, Canada; lcapretz@uwo.ca
[4] Department of Technical Informatics and Telecommunications, Gdańsk University of Technology, Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland
* Correspondence: abdullateef_16005851@utp.edu.my or balogun.ao1@unilorin.edu.ng

Abstract: As a result of the rapid advancement of mobile and internet technology, a plethora of new mobile security risks has recently emerged. Many techniques have been developed to address the risks associated with Android malware. The most extensively used method for identifying Android malware is signature-based detection. The drawback of this method, however, is that it is unable to detect unknown malware. As a consequence of this problem, machine learning (ML) methods for detecting and classifying malware applications were developed. The goal of conventional ML approaches is to improve classification accuracy. However, owing to imbalanced real-world datasets, the traditional classification algorithms perform poorly in detecting malicious apps. As a result, in this study, we developed a meta-learning approach based on the forest penalizing attribute (FPA) classification algorithm for detecting malware applications. In other words, with this research, we investigated how to improve Android malware detection by applying empirical analysis of FPA and its enhanced variants (Cas_FPA and RoF_FPA). The proposed FPA and its enhanced variants were tested using the Malgenome and Drebin Android malware datasets, which contain features gathered from both static and dynamic Android malware analysis. Furthermore, the findings obtained using the proposed technique were compared with baseline classifiers and existing malware detection methods to validate their effectiveness in detecting malware application families. Based on the findings, FPA outperforms the baseline classifiers and existing ML-based Android malware detection models in dealing with the unbalanced family categorization of Android malware apps, with an accuracy of 98.94% and an area under curve (AUC) value of 0.999. Hence, further development and deployment of FPA-based meta-learners for Android malware detection and other cybersecurity threats is recommended.

Keywords: android; malware detection; machine learning; meta-learner

Citation: Akintola, A.G.; Balogun, A.O.; Capretz, L.F.; Mojeed, H.A.; Basri, S.; Salihu, S.A.; Usman-Hamza, F.E.; Sadiku, P.O.; Balogun, G.B.; Alanamu, Z.O. Empirical Analysis of Forest Penalizing Attribute and Its Enhanced Variations for Android Malware Detection. *Appl. Sci.* **2022**, *12*, 4664. https://doi.org/10.3390/app12094664

Academic Editors: Lorenzo Musarella and Andrea Tundis

Received: 29 March 2022
Accepted: 4 May 2022
Published: 6 May 2022

1. Introduction

The emergence and rapid development of the internet and its technologies has led to a significant shift of human daily activities from conventional to automated or internet-based infrastructure or solutions [1,2]. In recent years, practically every member of society has used the internet in their daily activities, whether for social connection, information

gathering, health-related activities, or learning experiences [3,4]. The surge in internet usage has fuelled the expansion and popularity of mobile devices such as smartphones and tablets, as well as the Android operating system (OS), which is commonly found on such devices. The Android operating system, in particular, has emerged as the leading mobile OS, with a large global market value and penetration. More than a billion Android devices have been acquired, with Google Play alone responsible for an estimated 65 billion mobile software program downloads [5]. As a result, Android's growing popularity and usage, as well as the emergence of third-party app stores, have made it vulnerable to a wide range of malware [6].

Malware is a type of malicious software that is designed to execute damaging payloads on target devices, such as computers, cell phones, etc. [5,7]. In other words, malware refers to any software program that conducts undesired or suspicious activity on target devices [8]. Malware is classified into several categories, including viruses, worms, Trojans, rootkits, and ransomware. Malware variations are capable of effectively stealing private data, initiating distributed denial of service (DDoS) assaults, and causing disruption to such systems or devices. Each of these types of malware has the potential to be detrimental to mobile devices, which has created difficulty and worry in the field of information security [9,10].

According to the McAfee Labs Threats Report, the amount of mobile malware is constantly increasing [3,6,11,12]. In a similar study, Kaspersky [13] discovered more than five million harmful installation packages, including new Trojan and ransomware variants. Various strategies, including malware detection, vulnerability detection, and application reinforcement, have been suggested and implemented as feasible methods for protecting the Android OS [14–16]. Among the recommended security protection solutions for avoiding malicious apps, malware detection is one of the most common.

Malware detection methods are primarily classified into two types: signature-based and anomaly-based methods. To identify malicious behaviour, the signature-based strategy depends on a set of established features of such threats. Whereas this approach may detect previously known malware, it cannot detect new or unknown harmful behaviours [17–19]. In other words, the signature-based malware detection approach is incapable of detecting zero-day threats [20,21]. In contrast, the anomaly-based approach seeks to identify malicious network behaviour by continuously detecting any departures from known usual behaviour. Although anomaly-based approaches do not require much malware information, they are more successful at detecting previously unknown malware. Malware detection techniques based on machine learning (ML) techniques outperform classical (i.e., statistical and knowledge-based) solutions [15,22–24].

According to several studies, security experts and researchers are now focused on ML solutions for malware detection. The extraction of features from both malicious and benign Android applications is required for the deployment of ML methods for malware detection, and the resulting dataset is then utilized to train ML methods and produce malware detection models. Static and dynamic malware analyses are used to extract the features.

Static malware analysis examines the code of an Android application without executing it to find malicious patterns [9]. It is quite efficient at spotting malicious applications. The disadvantage of this approach is that it does not work against obfuscation tactics [3]. Dynamic malware analysis evaluates the characteristics of the application while it is operating in a simulated environment, such as a sandbox. This approach is effective, but it requires more resources and time, and it does not allow for the exploration of all execution routes. Using ML approaches and combining both of these strategies yields superior outcomes [11,12].

Nonetheless, while generating ML-based Android malware models, it is crucial to employ clean and well-defined datasets because the effectiveness of the ML model is heavily dependent on the quality of the dataset [25,26]. In particular, the distribution of class labels in a dataset is crucial for creating effective ML models. In real-world situations,

the distribution of class labels is uneven and, in many cases, extremely skewed. This innate tendency is known as the class imbalance problem [27,28].

The class imbalance problem occurs when one of the class labels in a dataset has a large number of examples (majority class), whereas the other class has a small number of instances (minority class) [29]. Insufficiently balanced class labels in a dataset make ML model classification very complicated and unreliable [27–29]. Because there are more instances of benign applications than malicious applications, Android malware detection is considered to have a class imbalance problem [3,16,20].

In this research work, adequate attention is given to the inherent imbalanced class labels while categorizing Android malware with high detection performance. Specifically, forest penalizing attributes (FPA) and its enhanced novel meta-learner variants are deployed for the detection of Android malware.

FPA builds extremely efficient decision trees by using the power of all attributes in a given dataset, using a weight assignment and weight increment approach. Specifically, FPA is a decision forest technique that constructs a series of highly accurate decision trees by using the strength of all attributes available in a dataset, as opposed to tree-based classifiers, which employ only a selection of non-class attributes. Simultaneously, to foster significant variety, FPA penalizes attributes that contributed to the most recent tree to construct subsequent trees.

In addition, enhanced variants of FPA based on cascade generalization and rotation forest meta-learners are proposed. These meta-learners are proposed to amplify the detection performance of FPA to generate a robust and generalizable Android malware detection model (Cas_FPA and RoF_FPA). In addition, the synthetic minority oversampling technique (SMOTE) is used as a data sampling method to alleviate the class imbalance problem in Android malware datasets.

The main objective of this research is to develop sophisticated ML-based (FPA and its variants) Android malware models in the presence of class imbalance problems in Android malware detection.

The main contributions of this research work are summarized as follows:

1. A detailed empirical analysis of the performance of FPA on balanced and imbalanced Android malware detection datasets;
2. The development of enhanced variants of FPA based on cascade generalization and rotation forest meta-learners;
3. An empirical evaluation and comparison of FPA and its enhanced variants with existing Android malware detection methods.

Furthermore, with this research work, we seek to provide answers to the following research questions (RQs):

1. How effective is the FPA algorithm in comparison to baseline classifiers in Android malware detection?
2. How effective are the enhanced variants of FPA (Cas_FPA and RoF_FPA) in Android malware detection?
3. How well do the proposed FPA and its variants perform as compared to current state-of-the-art methods in Android malware detection?

The remainder of the paper is organized as follows. An extensive review of existing Android malware detection models is presented in Section 2. Section 3 outlines the experimental methodology and elaborates on the proposed methods. Section 4 gives details on the experimental results, and Section 5 concludes the research.

2. Related Works

In this section, we examine and analyze current Android malware detection solutions built using various ML approaches.

Malware detection in Android devices and applications using ML algorithms has been extensively investigated in the literature. Many previous efforts in this research domain

have used baseline classifiers to detect Android malware. Sen, Aysan, and Clark [11] proposed SafeDroid, which employed a decision tree (DT) based on structural attributes instead of API calls and permissions, as is customary in conventional methods. The research findings showed that the technique based on structural traits detected new malware better than API-based features. However, the research work focused on structural-based features only. That is, the approach may not be effective for other families of Android malware. Moreover, SafeDroid likely inherits the instability and myopic (based on the greedy approach) problems of the DT algorithm [30]. Cen et al. [31] deployed a regularized logistic regression (RLR) for Android malware detection. From their results, it was observed that RLR had an F-1 score of 0.95. However, the problem of parameter tuning (penalty parameter λ) or RLR persists. Moreover, RLR still has issues with the collinearity of features. Fereidooni et al. [32] proposed Android Malware detection using STatic analySIs of Applications (ANASTASIA) for Android malware.

ANASTASIA is practically based on selected ML algorithms, such as support vector machine (SVM), logistic regression (LR), naïve Bayes (NB), random forest (RF), k-nearest neighbour (kNN), AdaBoost, XGBoost, and deep learning (DL). The suggested method outperformed some studied existing methods, but the huge overhead runtime cost is a drawback. The framework of ANASTASIA is complex and consists of ML methods with diverse computational characteristics, which invariably increase its computational complexity (time and space). In another context, Sahs and Khan [33] deployed an unsupervised one-class SVM for Android malware detection. The suggested method was used on benign instances only, which makes it somewhat inadequate for imbalanced classification. Additionally, Rathore et al. [34] proposed the use of clustering techniques for Android malware detection. Specifically, K-means, agglomerative, BIRCH, Gaussian mixture model, and DBSCAN clustering methods were utilized to group malware features based on their characteristics. However, finding or selecting an optimal number of clusters is a problem for the selected clustering methods.

In addition, some recent research has concentrated on the application of deep learning (DL) methods, such as long short-term memory (LSTM), deep convoluted neural network (DCNN), recurrent neural network (RNN), AlexNet, and Resnet [9,17,35–38]. Karbab, Debbabi, Derhab, and Mouheb [9] developed an automatic Android malware framework (MalDozer) based on an artificial neural network (ANN). MalDozer was trained to detect malware patterns based on collections of raw method calls in the assembly code. The proposed method recorded high-performance detection on the studied malware datasets; however, its performance in comparison with other existing Android malware methods were not included in the study. Aslan and Yilmaz [39] successfully hybridized AlexNet and Resnet-152 for Android malware detection. They integrated two broad pre-trained DL methods in an optimized manner. The performance of the proposed hybrid method was reported to be superior to that of its constituents (AlexNet and Resnet-152, as well as some existing Android malware detection methods). Nisa, Shah, Kanwal, Raza, Khan, Damaševičius, and Blažauskas [35] developed a feature-fusion solution to categorize Android malware. In particular, they augmented malware images to address any class imbalance that may exist and thereafter deployed a DCNN to generate a single-feature vector for Android malware classification. However, their findings lack generalizability, as the authors only focused on image-based malware, which is not the only form of Android malware. Additionally, Vinayakumar, Soman, Poornachandran, and Sachin Kumar [36] utilized LSTM, which is a form of RNN, for Android malware detection. A stacked LSTM with 32 memory blocks with a unit cell was used to detect applicable Android malware. The performance of the proposed LSTM with 1000 epochs at a 0.1 learning rate was superior to that of other studied baseline classifiers. Yadav et al. [40] conducted a comparative study of more than 20 CNN models on Android malware detection. Their reEfficientNet-B4 CNN-based model was suggested based on image malware representation. Although studies have shown that DL methods are somewhat gaining attention and, in some cases, perform better than con-

ventional ML methods, the issue of platform (hardware) dependence and hyperparameter tuning are some of their key limitations.

Some studies have proposed graph-based solutions for Android malware detection. Gao et al. [41] proposed Android malware detection (GDroid) based on a graph convolutional network (GCN). Their main approach is to align apps and APIs into a vast multigraph and transform the initial problem into a node classification problem. Likewise, Ou and Xu [42] developed a static-sensitive subgraph method that represents Android apps with high features. Their proposed method extends the function call graph (FCG) by adding sensitive nodes to it.

Significant attempts have also been made to improve the performance of baseline classifiers using ensemble techniques. Rahman and Saha [43] proposed StackDroid, a two-level architecture for detecting malware in Android smartphones that relies on stacking generalization to decrease the misdetection rate. The basic classifiers at the first level are extremely randomized tree (ERT), RF, multilayer perceptron (MLP), and stochastic gradient descent (SGD). By stacking the initial predictions of the baseline classifiers, the second level uses a meta-estimator/predictor (extreme gradient boosting, EGB) as the final predictor. StackDroid generated extremely encouraging results on the publicly accessible DREBIN dataset, with up to 97% AUC and 1.67 FPR values, respectively. Similarly, Yerima and Sezer [5] developed DroidFusion for detecting malware in mobile Android devices based on the prediction of fusion-selected baseline classifiers utilizing multiple classifier rank aggregation techniques. The performance was shown to be superior to that of previously employed stacked generalization techniques in multilevel architecture systems. In another study, Christiana et al. [44] used a voting ensemble of RF, SVM, and K-nearest neighbour (KNN) for Android malware detection. Their findings supported the voting ensemble's advantage over individual baseline classifiers in terms of sensitivity and accuracy. Additionally, Gupta and Rani [45] used several weighted voting and stacking ensemble algorithms for Android malware detection based on SVM, NB, decision table (DTable), RF, and kNN. Their experimental results showed that the studied baseline and stacking ensemble models are inferior to weighted voting ensemble approaches in term of performance. Collectively, a major drawback of these developed ensemble methods is their high computational cost. Furthermore, although these ensemble methods can guarantee high performance, they are hard to explain and interpret due to their black-box operation mechanism. In addition, ensemble methods have been reported to cope with imbalanced datasets, although ensembles are not viewed as a viable solution to the class imbalance problem [28].

Likewise, recent research has focused on the class imbalance problem in Android malware datasets. Oak et al. [46] developed an activity-sequencing model to improve the performance of Android malware detection systems by removing the influence of an imbalanced dataset. They used a bidirectional encoder representation from transformers (BERT) to learn the sequence of dynamic actions on a highly skewed dataset of android malware. The technique was found to be successful in managing highly unbalanced android malware datasets, with a 0.919 F1 score Also, Xu et al. [47] developed a fuzzy–synthetic minority oversampling strategy to address imbalanced datasets in Android malware detection. The authors presented cost-sensitive forest (CS-Forest) as a method to overcome the data imbalance problem in android malware detection The experimental results indicated that CS-Forest was superior to similar systems without cost-sensitive methodologies and outperformed SMOTE and borderline SMOTE techniques. However, a limitation of CS-Forest is the allocation of misclassification cost (performed manually) and representation of the cost matrix. In another study, to address the class imbalance problem in Android malware detection, Dehkordy and Rasoolzadegan [48] successfully hybridized undersampling with SMOTE. ML methods such as KNN, SVM, and Iterative Dichotomiser 3 (ID3) were deployed on the balanced dataset in this study. The proposed technique yielded an encouraging detection result, with detection accuracy as high as 97%.

Summarily, as shown in Table 1, several solutions have been proposed, from simple baseline classifiers to sophisticated solutions, such as cost-sensitive learning and DL methods. However, the development of better approaches for Android malware detection is ongoing, as novel methods are required to deal with the dynamism of Android malware. Additionally, the preceding reviews have shown that class imbalance is another issue that affects the performance of Android malware detection models. As a solution, we propose a decision forests algorithm known as forest penalizing attributes (FPA) and its enhanced meta-learner variations (Cas_FPA and RoF_FPA) for Android malware detection. Furthermore, SMOTE data sampling is incorporated to alleviate the inherent class imbalance problem that may be present in the Android malware datasets.

Table 1. Summary of key-related studies.

References	Dataset	Technique	Class Imbalance	Limitations
Sen, Aysan and Clark [11]	Drebin and Malgenome	SafeDroid: decision tree (DT)	Not applicable	Static Features only
Cen, Gates, Si and Li [31]	Market Dataset	probabilistic discriminative model (regularized logistic regression)	Not applicable	Parameter Tuning and Collinearity Issue
Fereidooni, Conti, Yao and Sperduti [32]	Drebin and Malgenome	SVM, LR, NB, RF, kNN, AdaBoost, XGBoost, and deep learning (DL	Not applicable	High over-head runtime cost
Sahs and Khan [33]	Private Dataset	One-SVM	Not applicable	Inadequate for class imbalance classification tasks
Rathore, Sahay, Chaturvedi and Sewak [34]	Drebin and Malgenome	Clustering techniques	Not applicable	The optimum number of clusters issue
Karbab, Debbabi, Derhab and Mouheb [9]	Drebin and Malgenome	MalDozer: ANN	Not applicable	Not stated
Aslan and Yilmaz [39]	Malimg, MicrosoftBig2015 and Malevis	Hybrid AlexNet and Resnet-152	Not applicable	Suitable for image-based malware only
Nisa, Shah, Kanwal, Raza, Khan, Damaševičius and Blažauskas [35]	Malimg	Feature fusion based on DCNN	Data augmentation	Suitable for image-based malware only
Vinayakumar, Soman, Poornachandran and Sachin Kumar [36]	Not Stated	LSTM	Not applicable	Platform Dependence and parameter tuning
Yadav, Menon, Ravi, Vishvanathan and Pham [40]	R2-D2	CNN	Not applicable	Platform Dependence and parameter tuning
Gao, Cheng and Zhang [41]	AMD, Drebin and Malgenome	GDroid: graph convolution network	Not applicable	Suitable for image-based malware only
Ou and Xu [42]	Market Dataset	Sensitive function call graph (SFCG)	Not applicable	Continuous configuration to generate complete sensitive APIs
Rahman and Saha [43]	Drebin	StackDroid: stacking ensemble based on ERT, RF, MLP, and SGD	Not applicable	High computational cost
Yerima and Sezer [5]	Drebin, Malgenome, and McAfee	DroidFusion: multilevel weight ranking-based approach	Not applicable	High computational cost
Christiana, Gyunka and Oluwatobi [44]	Private Dataset	Voting ensemble based on RF, SVM, and kNN	Not applicable	High computational cost

Table 1. *Cont.*

References	Dataset	Technique	Class Imbalance	Limitations
Gupta and Rani [45]	Windows OS Dataset	Stacking and voting ensemble based on SVM, NB, decision table (DTable), RF, and kNN	Not applicable	High computational cost
Oak, Du, Yan, Takawale and Amit [46]	Wildfire Dataset	BERT	Sequence modeling	Only applicable to sequence datasets
Xu, Wu, Zheng, Niu and Yang [47]	UNB ISCX	CS-Forest	Fuzzy–synthetic minority oversampling strategy	Allocation of misclassification cost (done manually) and representation of cost matrix
Dehkordy and Rasoolzadegan [48]	AMD and Drebin	ID3, kNN, and SVM	Hybrid Undersampling and SMOTE	Parameter tuning of kNN and SVM

3. Methodology

In this section, we describe the research approach that was used in this study, specifically detailing the proposed FPA and its variants. Additionally, the performance evaluation metrics and the experimental procedure for the Android malware datasets are illustrated.

3.1. Forest Penalizing Attribute (FPA) Algorithm

The forest penalizing attribute (FPA) algorithm, as defined by Adnan and Islam [49], fosters significant variety by taking weight-related concepts into consideration, which include but are not limited to weight assignment strategy and weight increment strategy. FPA is a strategy that typically constructs a collection of highly accurate decision trees by using the strength found in all non-class characteristics accessible in the supplied dataset. Algorithmically, FPA changes the weights of features that occur in the most recent tree at random within a weight range (WR) set as:

$$WR^{\theta} = \begin{cases} [\mathbf{0.0000}, e^{-1/\theta}], \theta = \mathbf{1} \\ [e^{-1/\theta - 1} + \delta, e^{-1/\theta}], \theta > 1 \end{cases} \quad (1)$$

where θ denotes the feature level, and δ enforces non-overlapping of WR for different levels. In light of resolving the adverse influence of maintaining weights that are missing in the latest tree, FPA introduces a general increase in weights of the feature that has not been evaluated in the succeeding trees. The reason for this is that a lower-level feature may impact more logic rules than a higher-level feature. Consequently, to uncover a varied collection of logic rules, features checked at lower levels are meant to be avoided more carefully in a prospective tree than those assessed at higher levels. Additionally, to enhance the likelihood of having various weights among features in the same level, FPA chooses an attribute's weight at random from the weight range assigned to the attribute's level.

In comparison to the random forest (RF), random subspace (RS), and extremely randomized trees (ERT), FPA aims to ensure that high-performing attributes/features/nodes are selected or maintained on the ensuing tree. RF, RS, and ERT are primarily based on the random feature weight mechanism that randomly generates and assigns weights, which often leads to a mismatch in the weights of attributes. Consequently, there is no guarantee that high-performing attributes/features/nodes will be regularly selected or maintained. However, FPA introduces an exponent (known as p value) to the weight values to enhance the impact of the weights [50]. Additionally, unlike RS and ERT, FPA does not deploy subspacing of feature sets, which often leads to irregular performances for low- and high-dimensional datasets. Ultimately, FPA is more suitable for ML processes. To the best of our knowledge, FPA has not been used in the context of Android malware detection.

3.2. Meta-Learners

3.2.1. Cascade Generalization

Cascade generalization is a fast method that works at both the base and meta levels. A meta-level classifier trains a model directly from the projections of base-level classifiers in this stage to evaluate the eventual ensemble result [51]. The predictions of the base-level classifier (in this case, FPA) are used to expand the dimensionality of the input space in cascade generalization. This is achieved by attaching the output of FPA to each training sample as a new function. The corollary of this is that the initial input features are used by FPA and meta-level classifiers, with the meta-level classifiers having access to extra characteristics (predictions from the FPA classifier) [51]. The pseudocode for the Cas_FPA approach is shown in Algorithm 1.

Algorithm 1: Cascade Generalization based Forest Penalizing Attributes (Cas_FPA).

Input: The training set, S = $\{x_i, y_i\}$, $i = 1 \ldots m$, $y_i \in Y$, $Y = \{c_1, c_2, \ldots, c_k\}$, c_k, is the class label;
The number of Iterations, I = 100;
U:= Initial unlabelled instances
L:= Initial labelled instances
A:= Acceptance Threshold
M_i:= Instance with Most Confident Prediction
Base Learner B:= FPA
1 Initializing weights distribution of $D_1(i) = 1/m$
2 Train FPA as base learner B
3 For i = 1 to I
4 For each iteration, deploy FPA to identify instances with MCP
5 Remove M_i from U and append to Y
6 Retrain *FPA* as B on new L
Output: Predicted class label

3.2.2. Rotation Forest

The rotation forest (RoF) meta-learner uses feature extraction to construct classifier models. By randomly partitioning the feature set into N subsets, RF generates training data for a baseline learner, and principal component analysis (PCA) is used for each of the produced subsets [52,53]. All primary components are preserved to maintain the variability in the data. As a result, N axis rotations occur to provide new features for the baseline learner (in this case, FPA). The rotation's essence is to allow for contemporaneous independent accuracy and variety within the ensemble. FPA's diversity is achieved by feature extraction. In Algorithm 2, the proposed RoF_FPA algorithm is provided on the premise that X is the training dataset, Y is the class label, and F is the feature sets.

Algorithm 2: Rotation Forest-based Forest Penalizing Attribute (RoF_FPA) Algorithm.

Input: The training set, $X = \{x_i, y_i\}$, $i = 1 \ldots m$, $y_i \in Y$, $Y = \{c_1, c_2, \ldots, c_k\}$, c_k, is the class label;
Baseline Learner: *FPA*

1. Choose a value for K that is a factor of n; let F be randomly divided into K parts of the distinct subsets; each subset must contain $N = n/k$ number of features.
2. Select the corresponding columns of attributes in the subset, $T_{i,j}$, from the training dataset, X; then, form a new matrix, $X_{i,j}$. Extract a bootstrap subset of objects 3/4 of X to make a new training dataset, $X'_{i,j}$
3. Use Matrix $X'_{i,j}$ as feature transform to produce the coefficient in the matrix, $P_{i,j}$, in which the jth column coefficient is the characteristic component, jth.
4. Construct a sparse rotation matrix, S_i using the coefficient obtained in the matrix, $P_{i,j}$.

Output: classifier T_i of $d_{i,j}\left(XS_i^{\mathrm{f}}\right)$ to determine x belongs to the class y_i.
Then, calculate class confidence:

$$\alpha_j(x) = \frac{1}{L} \sum_{i=1}^{L} d_{i,j}\left(XS_i^{\mathrm{f}}\right)$$

Assign the category with the largest $\alpha_j(x)$ value to x.

3.3. Experimental Framework

In this section, we review the experimental procedure that was used in this study (Figure 1). The procedure was designed to experimentally evaluate and validate the effectiveness of FPA and its enhanced variants for detecting Android malware. Specifically, two scenarios were evaluated and explored, and the resultant Android detection model performances were compared in a non-biased and consistent manner.

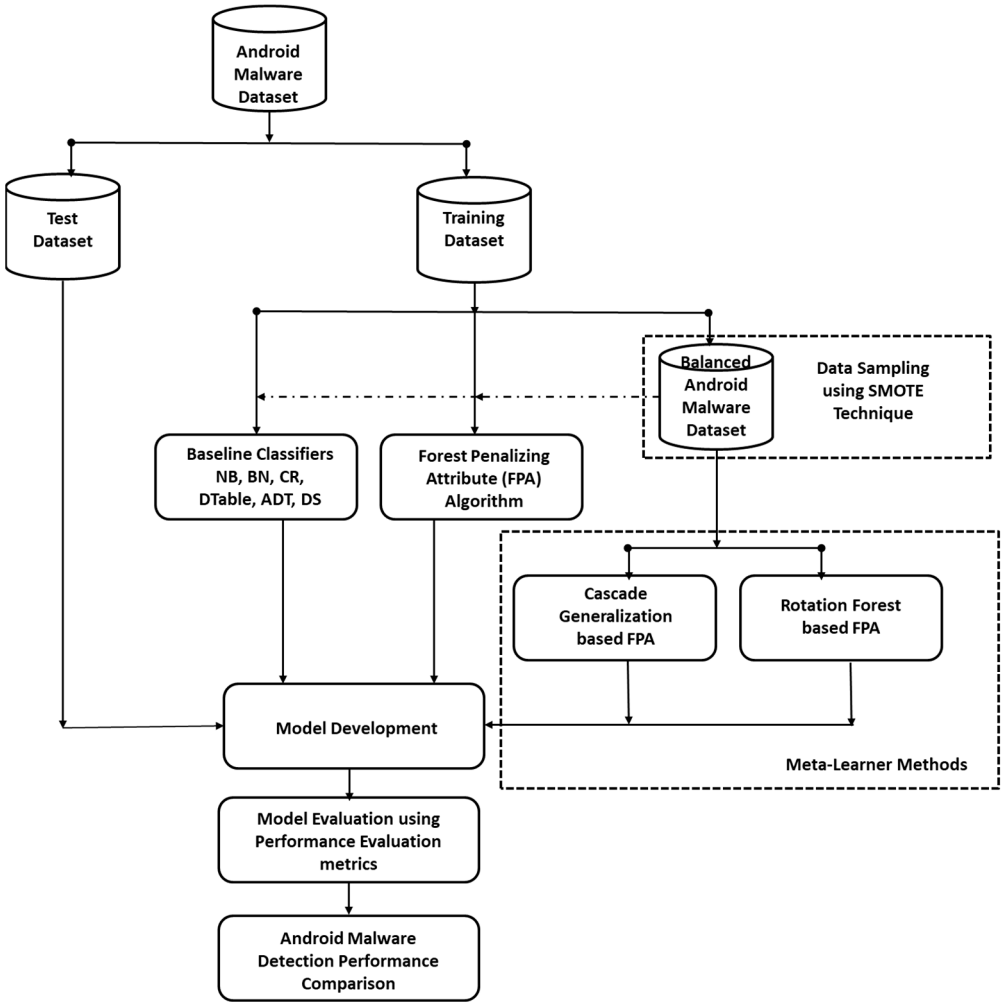

Figure 1. Experimental Framework.

Scenario 1: First, FPA and selected ML algorithms with varying computational characteristics were specifically applied to the original Android malware datasets. The result of this experiment will empirically validate FPA's performance and efficacy in detecting Android malware. Furthermore, to resolve the inherent class imbalance in the Android malware datasets, a data sampling technique was deployed. In other words, data over-sampling (SMOTE) was employed to resolve the class imbalance in the malware datasets. SMOTE is a notable data sampling approach used to solve the issue of class imbalance [27,28]. The results of these tests will show the influence of the data sampling approach in FPA in Android malware detection.

Scenario 2: In this case, FPA and its enhanced variants were applied to the balanced datasets. The enhanced variants of FPA are the cascade generalization-based FPA (Cas_FPA) and rotation forest-based FPA (RoF_FPA). The performance of FPA and its variants were analysed and compared with existing Android malware detection models.

The observed findings and conclusions derived from the experimental results (Scenarios 1 and 2) were utilized to address the research questions listed in Section 1.

For training and testing models based on the scenarios, the datasets were divided into test (30%) and train datasets (70%). Thereafter, the training datasets were balanced using SMOTE and later used to train the experimented models using the 10-fold cross-validation (CV) technique. The K-fold (where k = 10) CV approach was used for the development and evaluation of the Android malware detection models. The 10-fold CV option is based on the ability to create Android malware detection models with a low impact on the issue of class imbalance [28,54–57]. The training and testing datasets were randomly generated, with no duplicates or shared values. Eventually, for each studied dataset, the average of the returned outcomes was utilized as the final assessment criterion. In addition, each experimental process was repeated 10 times; that is, each experiment was conducted 100 times to avoid bias [58–60]. The Waikato Environment for Knowledge Analysis (WEKA) machine learning library [61] and R programming language [62] were used for the experimentation on an Intel(R) Core™ machine equipped with i7-6700, running at a speed of 3.4 GHz CPU with 16 GB RAM.

3.4. Android Malware Datasets

Two Android malware datasets (Drebin and Malgenome) were used in the experimentation phase of this study. These datasets are publicly accessible and are often used in contemporary Android malware studies [9,63–66].

The Drebin malware dataset, which was collected between August 2010 and October 2012, is composed of a total of 15,036 samples, of which 5560 are malware and 9476 are benign, with 215 distinct features. The 5560 malware instances contain samples from 179 different malware families [5]. Additionally, Drebin is made up of 53% manifest permission and 33% API signature, whereas other forms of API call signature (such as intent and command signature) comprise 14% of the dataset [67]. Table 2 presents the top 10 families of malware present in the Drebin dataset.

Table 2. Top 10 malware families in the Drebin dataset.

Malware Family	Number of Instances
Fake Installer	925
DroidKungFu	667
Plankton	625
Opfake	613
Ginmaster	339
BaseBridge	330
Iconosys	152
KMin	147
FaceDoc	132
Geinimi	92

Concerning the Malgenome dataset, it has 3799 samples, of which 1260 are malware and 2539 are benign. This malware was classified methodically based on factors such as installation techniques, activation mechanisms, and the type of malicious content conveyed. According to research, this malware evolves quickly to avoid detection by conventional mobile antivirus software [68]. Similar to Drebin, Malgenome contains 215 features derived from 49 different Android malware families [5]. Table 3 presents the top 10 Android malware families present in the Malgenome dataset.

Extended details on the Drebin and Malgenome Android malware datasets are publicly available online [5,67,68].

Table 4 illustrates the characteristics of the experimented Drebin and Malgenome Android malware datasets, and Table 5 presents the prominent feature names and categories present in both datasets.

Table 3. Top 10 malware families in the Malgenome dataset.

Malware Family	Number of Instances
DroidKungFu3	309
AnserverBot	187
BaseBridge	122
DroidKungFu4	96
Geinimi	69
Pjapps	58
KMin	52
GoldDream	47
DroidDreamLight	46
DroidKungFu3	30

Table 4. Details of Android malware datasets.

Algorithm	Number of Features	Number of Instances	Number of Benign	Number of Malware
Drebin	215	15,036	9476	5560
Malgenome	215	3.799	2539	1260

Table 5. Feature details of malware in Malgenome and Drebin datasets.

Feature Name	Feature Category	Number of Features
WRITE_SETTINGS SET_TIME ADD_VOICEMAIL . . .	Manifest Permission	113
URL ClassLoader PathClassLoader . . .	API Calls	73
Remount Chown	Commands	6
Intent.action.BOOT_COMPLETED Intent.action.TIME_SET Intent.action.Action_SHUTDOWN	Intents	23

3.5. Performance Evaluation Metrics

In this study, accuracy, F-measure, and area under the curve (AUC) were used to assess the detection performances of Android malware models. The selection of these assessment measures was based on current research, which shows widespread and regular use of these evaluation metrics for Android malware detection.

i. Accuracy measures the overall rate at which the actual labels of all instances are correctly predicted [69]. It is calculated using Equation (2):

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{2}$$

ii. F-measure is the weighted average of both the recall (R) and precision (P) metrics. It emphasizes how good a classifier is at maximizing both the precision and recall values simultaneously. F-measure can be computed as defined in Equation (3):

$$\text{Fmeasure} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \tag{3}$$

iii. The area under the curve (AUC) plots the FP rate on the *X*-axis and the TP rate on the *Y*-axis. AUC is not susceptible to majority class bias and does not ignore the minority class during its evaluation.

4. Results and Discussion

In this section, we present and discuss our results based on the experimental framework as outlined in Section 3.3. Tables 6 and 7 compare the detection performances of FPA against baseline classifiers (NB, BN, conjunctive rule (CR), decision table (DTable), alternating decision tree (ADT), and decision stump (DS)) on the original Android malware datasets. It should be noted that the selection of the baseline classifiers was based on their diverse computational characteristics and performance as reported in existing studies. Furthermore, Tables 8 and 9 present the detection performances of FPA against the baseline classifiers on SMOTE-balanced Android malware datasets. The purpose of this is to showcase the effect of the data sampling method (SMOTE) on the performance of FPA. Tables 10 and 11 present analyses of the detection performance of FPA and its enhanced variants (Cas_FPA and RoF_FPA) on the SMOTE-balanced Android malware datasets. Lastly, the detection performance of FPA, Cas_FPA, RoF_FPA, Cas_FPA+SMOTE, and RoF_FPA+SMOTE was compared with existing state-of-the-art Android malware models. This section also includes some figures to illustrate the significance of the research findings. The top results are highlighted in bold, whereas the suggested methods are denoted by an asterisk.

Table 6. Detection performances of FPA and baseline classifiers on the Malgenome dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
NB	92.58	0.926	0.926	0.926	0.926	0.926	0.048
BN	92.73	0.927	0.927	0.927	0.927	0.927	0.046
CR	75.44	0.754	0.754	0.754	0.754	0.754	0.139
DTable	91.81	0.911	0.911	0.911	0.911	0.938	0.066
ADT	95.89	0.991	0.959	0.959	0.959	0.959	0.056
DS	74.65	0.795	0.754	0.813	0.747	0.747	0.177
* FPA	**98.94**	**0.998**	**0.989**	**0.989**	**0.989**	**0.989**	**0.016**

Table 7. Detection performances of FPA and baseline classifiers on the Drebin dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
NB	82.42	0.824	0.824	0.824	0.824	0.824	0.125
BN	82.78	0.828	0.828	0.828	0.828	0.828	0.124
CR	75.55	0.756	0.758	0.833	0.756	0.756	0.159
DTable	90.2	0.922	0.922	0.924	0.922	0.922	0.077
ADT	93.73	0.981	0.937	0.937	0.937	0.937	0.063
DS	75.31	0.787	0.756	0.823	0.753	0.753	0.169
* FPA	**98.13**	**0.997**	**0.981**	**0.981**	**0.981**	**0.981**	**0.025**

Table 8. Detection performances of FPA and baseline classifiers on the SMOTE-balanced Malgenome dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
NB	95.26	0.994	0.953	0.954	0.954	0.953	0.047
BN	94.50	0.994	0.945	0.948	0.945	0.945	0.055
CR	81.42	0.809	0.809	0.857	0.814	0.814	0.185
DTable	93.64	0.986	0.936	0.937	0.936	0.936	0.063
ADT	96.05	0.993	0.960	0.961	0.960	0.960	0.040
DS	78.88	0.796	0.794	0.828	0.800	0.800	0.200
* FPA	**98.85**	**0.999**	**0.989**	**0.989**	**0.989**	**0.989**	**0.011**

Table 9. Detection performances of FPA and baseline classifiers on the balanced (SMOTE) Drebin dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
NB	85.61	0.945	0.945	0.945	0.945	0.856	0.144
BN	85.58	0.942	0.942	0.942	0.942	0.856	0.144
CR	80.10	0.840	0.840	0.840	0.801	0.801	0.199
DTable	92.41	0.976	0.976	0.976	0.976	0.976	0.024
ADT	93.73	0.981	0.937	0.937	0.937	0.937	0.063
DS	79.44	0.787	0.79	0.825	0.794	0.794	0.205
* FPA	**98.37**	**0.997**	**0.984**	**0.984**	**0.984**	**0.984**	**0.016**

Table 10. Detection performances of FPA, Cas_FPA, and RoF_FPA on the Malgenome dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA	98.94	0.998	0.989	0.989	0.989	0.989	0.016
* Cas_FPA	**99.45**	**1**	**0.994**	**0.994**	**0.994**	**0.994**	**0.008**
* RoF_FPA	99.00	0.999	0.990	0.990	0.990	0.990	0.018

Table 11. Detection performances of FPA, Cas_FPA, and RoF_FPA on the Drebin dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA	98.13	0.997	0.981	0.981	0.981	0.981	0.025
* Cas_FPA	**98.83**	**0.999**	**0.988**	**0.988**	**0.988**	**0.988**	**0.016**
* RoF_FPA	98.38	0.997	0.984	0.984	0.984	0.984	0.023

4.1. Android Malware Detection Performance Comparision of FPA and Baseline Classifiers

In this section, the Android malware detection performance of FPA is compared with that of the selected baseline classifiers on original and SMOTE-balanced Malgenome and Drebin datasets, as depicted in Section 3.3 (Scenario 1).

Table 6 presents the detection performance of FPA and the selected baseline classifiers on the Malgenome dataset. FPA outperformed the baseline classifiers based on the evaluation metrics that were considered. Specifically, based on detection accuracy values, FPA had a 98.94% accuracy value, which is +3.15% higher than that of the best-performing base classifier, ADT, with a 95.89% detection accuracy. Just below ADT, the Bayesian models (NB and BN) also recorded good detection accuracy values of more than 92% detection accuracy. The high detection accuracy value achieved by FPA, even on the unbalanced Malgenome dataset, emphasizes its robustness and efficacy for Android malware detection. FPA achieved the highest AUC value, at 0.998, followed closely by ADT (0.991). In addition, FPA achieved a very good balance between precision and recall, recording an F-measure value of 0.989. This observation further affirms the detection performance consistency of FPA as compared to other experimented baseline classifiers. Although ADT had a good F-measure value, it is still inferior to FPA (+3.12%). Considering the misclassification rate, FPA obtained the lowest score in FPR, with only a 0.016 chance of misclassifying any instance. This very low FPR shows that FPA is not prone to the misclassification errors often observed in other base classifiers. It can also be observed that the Bayesian models (NB and BN) outperformed the rest of the base classifiers in terms of FPR. Figure 2 presents a graphical representation of the performance comparison.

Similar performance patterns were observed on the Drebin dataset (Table 7), as FPA recorded superior detection performances based on accuracy (98.13%), AUC (0.997), F-measure (0.981), and FPR (0.025) values when compared with the baseline classifiers. It was also observed that the detection performances of the classifiers on the Malgenome dataset are somewhat better than those on the Drebin dataset. Among the base classifiers, ADT recorded the highest detection accuracy (93.73%), AUC (0.981), and F-measure (0.937) values and the lowest FPR (0.063). It can also be observed that DTable closely follows

the performance of ADT concerning accuracy (90.20%), AUC (0.922), F-measure (0.922), and FPR (0.077) values. That is, DTable showed a better performance than the Bayesian algorithms (BN and NB) on the Drebin dataset in contrast to the Malgenome dataset. Generally, all the algorithms achieved good performance on the original datasets, except CR and DS, the performances of which were relatively low. Figure 3 shows the Android malware detection performances of FPA and the baseline classifiers on the Drebin dataset.

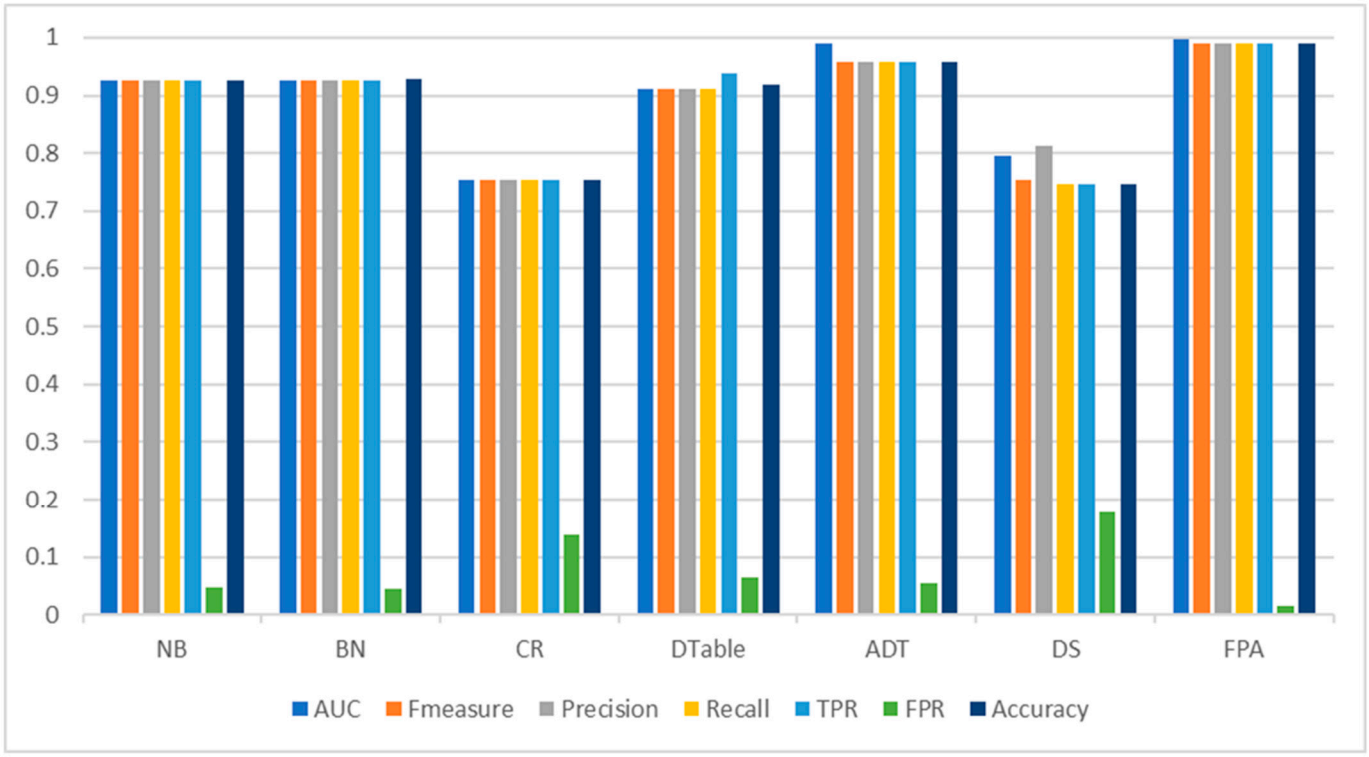

Figure 2. Android malware detection performance of FPA and baseline classifiers on the Malgenome dataset.

We then proceeded to investigate the performance of FPA and baseline classifiers on SMOTE-balanced Android malware datasets. The essence of the deployment of the SMOTE data sampling technique is to remove the inherent class imbalance problem in the Android malware datasets. Moreover, the preference for SMOTE is based on its usage and efficacy, as reported in existing studies. Tables 8 and 9 present the experimental results of FPA and the baseline classifiers on the SMOTE-balanced Malgenome and Drebin datasets, respectively.

As presented in Table 8, FPA still outperformed the baseline classifiers on all performance metrics evaluated. Specifically, FPA had a detection accuracy value of 98.85% and an FPR as low as 0.011. However, based on evaluation metrics (AUC and F-measure), the detection performance of the baseline classifiers was comparable to FPA. This observation can be attributed to the deployment of data sampling (SMOTE) to address the class imbalance problem. That is, the detection performances of the classifiers evaluated in the experiment improved as compared to their respective detection performances on the original Malgenome dataset. In particular, CR (+7.9%) and DS (+5.7%) achieved the greatest improvement in accuracy values. Additionally, a notable improvement in accuracy was observed in NB (+2.9%), BN (+1.9%), and DTable (+2.0%). ADT (+0.2%) recorded a slight improvement in its detection accuracy. Regardless, the detection accuracy performance of FPA is still superior. Concerning AUC values, DTable (+8.2%) achieved the greatest improvement, closely followed by NB (+7.34%), CR (+7.29%) and BN (+7.22%). Slight improvements were noticed in the AUC values of ADT (+0.20%), DS (+0.12%), and FPA (+0.10). A similar occurrence was observed with F-measure, as CR (+7.29%), DS (+5.31%),

NB (2.92%), DTable (2.74%), and BN (1.94%) had improved F-measure values. However, concerning the FPR metric, FPA (−31.25%) achieved the greatest improvement, with a reduction of 31.25% in FPR value. It should be noted that the lower the FPR value, the better the detection performance. Similarly, ADT (28.57%), DTable (4.55%), and NB (2.08%) exhibited a relative reduction in their FPR values. In general, the detection performances of FPA and the baseline classifiers improved on the SMOTE-balanced Malgenome dataset, but FPA still achieved the best overall performance. Figure 4 illustrates the Android malware detection performances of FPA and the baseline classifiers on the SMOTE-balanced Malgenome dataset.

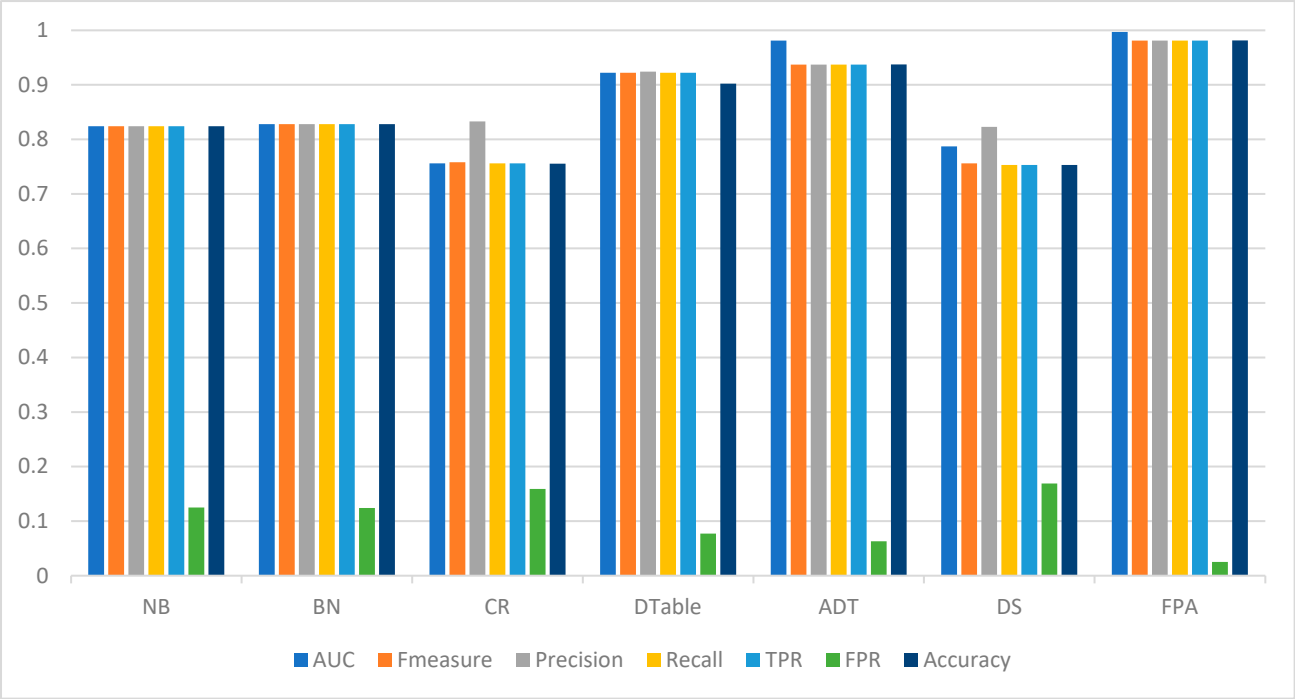

Figure 3. Android malware detection performance of FPA and baseline classifiers on the Drebin dataset.

Furthermore, on the SMOTE-balanced Drebin dataset (Table 9), FPA still achieved the highest detection performance results in all performance measures considered. A similar finding to that observed in the experimental results on the SMOTE-balanced Malgenome dataset was also observed in the SMOTE-balanced Drebin dataset. That is, all classifiers recorded an improvement in their respective detection performance. As shown in Table 5, FPA (+0.24%), CR (+6.02%), DS (+5.48%), NB (+3.87%), BN (+3.38%), and DTable (+2.45%) had improved accuracy values when compared to their respective detection performances on the original Drebin dataset. Additionally, concerning AUC values, the Bayesian models (NB (+14%), BN (+14%), CR (+11.11%), and DTable (+5.86%)) each recorded significant improvement. The analysis based on the F-measure metric showed similar findings. Both FPA and the base classifiers had improved F-measure values, except those of ADT, for which results remain unchanged. FPA and DTable had a 36.00% and 68.83% reduction in FPR values, respectively. Other classifiers (NB, BN, DS, ADT, and CR) also had an increased FPR value. Figure 5 graphically represents the detection performances of FPA and baseline classifiers on the SMOTE-balanced Drebin dataset.

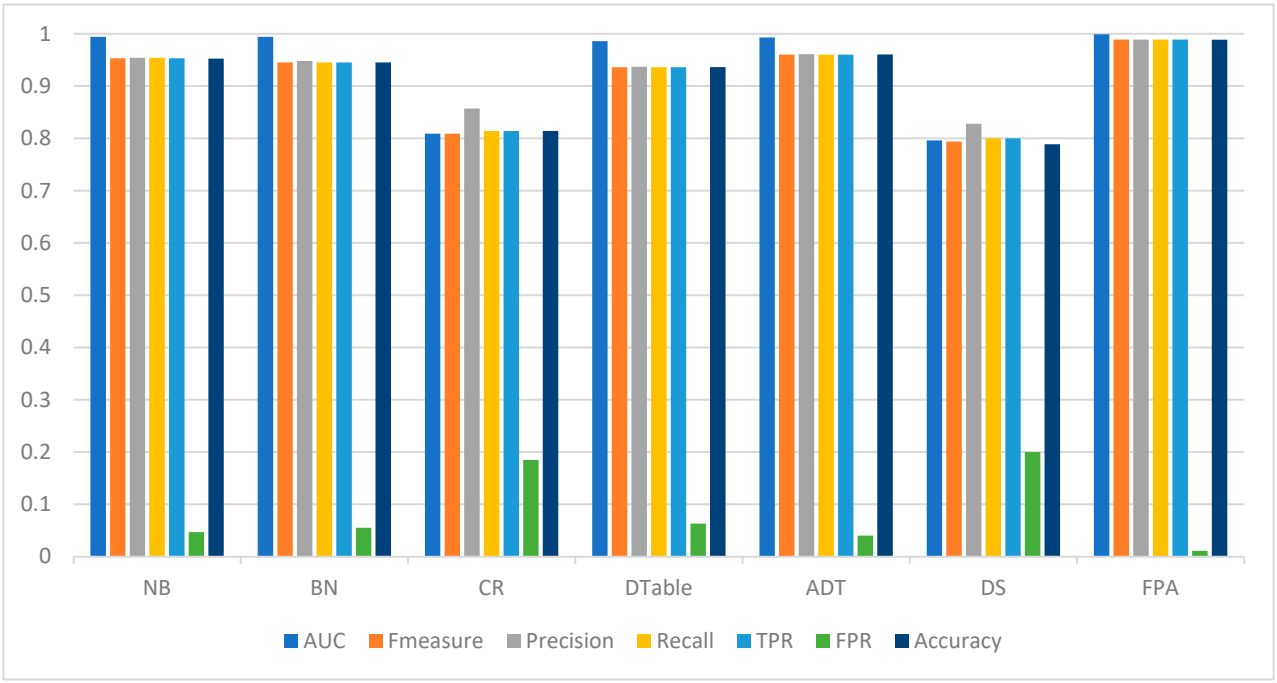

Figure 4. Android malware detection performance of FPA and baseline classifiers on the SMOTE-balanced Malgenome dataset.

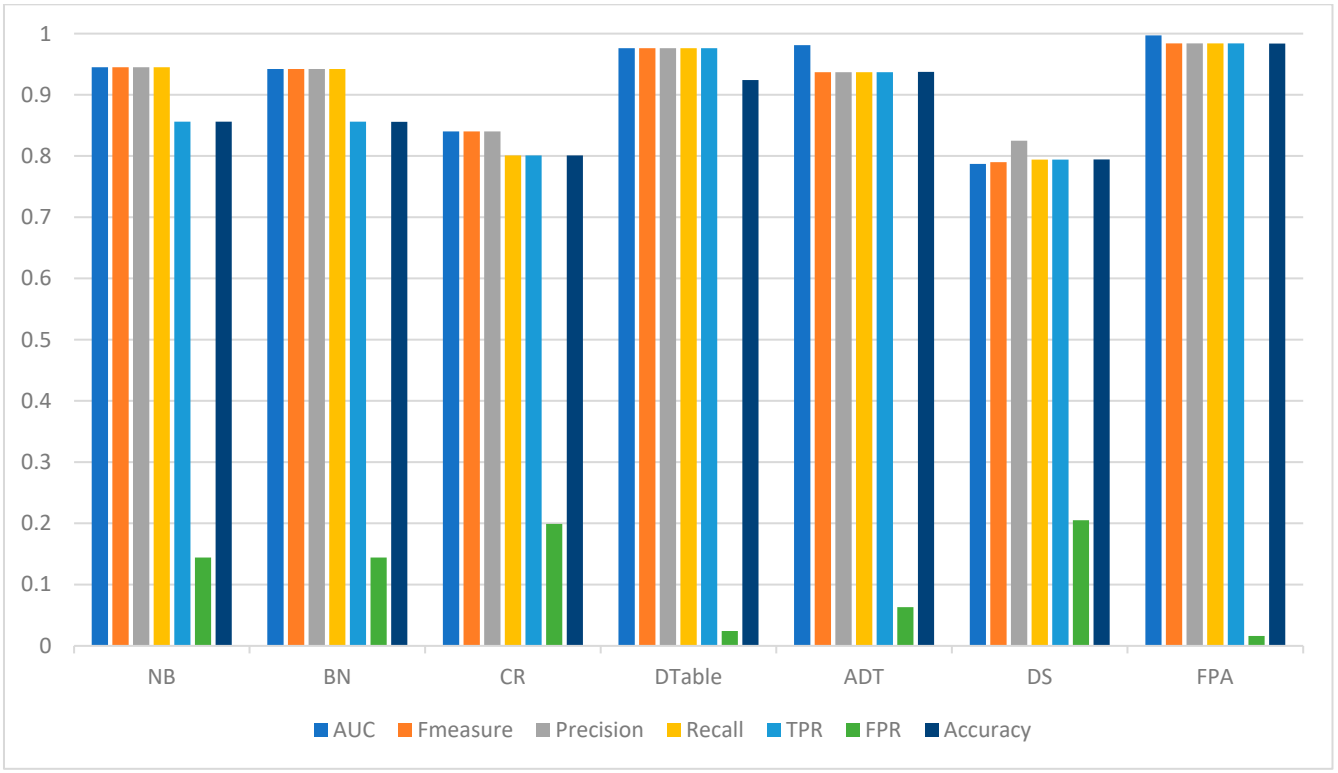

Figure 5. Android malware detection performance of FPA and baseline classifiers on the SMOTE-balanced Drebin dataset.

Based on the foregoing experimental results on the Malgenome and Drebin datasets, the following findings were observed:

1. FPA recorded a higher performance than the baseline classifiers for Android malware detection. The baseline classifiers were selected based on their usage and performances, as reported in existing studies.
2. The deployment of a data sampling method (in this case, SMOTE) not only alleviated the class imbalance problem but also improved the detection performances of the FPA and baseline classifiers.
3. FPA can perform well for Android malware detection with or without the application of a data sampling method.

These findings authenticate and validate the selection of FPA for Android malware detection. However, to amplify the FPA detection performances, FPA variants based on meta-learning (Cas_FPA and RoF_FPA) concepts were developed. Empirical analysis of the experimental results is presented in Section 4.2.

4.2. Android Malware Detection Performance Comparison of FPA, Cas_FPA and RoF_FPA

In this section, the Android malware detection performance of FPA is compared with that of its enhanced variants (Cas_FPA and RoF_FPA) on original and SMOTE-balanced Malgenome and Drebin datasets, as shown in Tables 10–13, respectively. Thereafter, the detection performance of FPA, Cas_FPA, RoF_FPA, Cas_FPA+SMOTE, and RoF_FPA+SMOTE was compared with that of existing state-of-the-art Android malware detection models (Tables 14 and 15), as described in Section 3.3 (Scenario 2).

Table 12. Detection performance of FPA, Cas_FPA, and RoF_FPA on the SMOTE-balanced Malgenome dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA+SMOTE	98.85	0.999	0.989	0.989	0.989	0.989	0.011
* Cas_FPA+SMOTE	**99.42**	**1**	**0.994**	**0.994**	**0.994**	**0.994**	**0.006**
* RoF_FPA+SMOTE	99.07	**1**	0.991	0.991	0.991	0.991	0.009

Table 13. Detection performance of FPA, Cas_FPA, and RoF_FPA on the SMOTE-balanced Drebin dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA+SMOTE	98.37	0.997	0.984	0.984	0.984	0.984	0.016
* Cas_FPA+SMOTE	**98.94**	**0.999**	**0.989**	**0.990**	**0.989**	**0.989**	**0.011**
* RoF_FPA+SMOTE	98.49	0.998	0.985	0.985	0.985	0.985	0.015

Tables 10 and 11 present the detection performances of FPA, Cas_FPA, and RoF_FPA on the Malgenome and Drebin datasets. On the Malgenome dataset (Table 10), both Cas_FPA and RoF_FPA recorded superior detection performance over FPA. Specifically, Cas_FPA achieved a detection accuracy value of 99.45%, an AUC value of 1, an F-measure value of 0.994, and an FPR value of 0.008. RoF_FPA had a similar performance, with a detection accuracy value of 99%, an AUC value of 0.999, an F-measure value of 0.990, and an FPR value of 0.018. Notably, the AUC values of 1 and 0.999 recorded by Cas_FPA and RoF_FPA, respectively, indicate the effectiveness of the two models (Cas_FPA and RoF_FPA) in distinguishing Android malware applications from benign applications with approximately 100% certainty. Additionally, a very low FPR of almost zero recorded by Cas_FPA (0.008) showed its reliable counteraction towards false Android malware detection. Furthermore, on the Drebin dataset (Table 11), Cas_FPA and RoF_FPA outperformed FPA in terms of evaluation metrics. Cas_FPA (98.83%) and RoF_FPA (98.38%) showed a +0.71% and +0.25% increment, respectively, in detection accuracy value over FPA (98.13%). Concerning FPR value, significant improvements were observed; Cas_FPA (0.016) and RoF_FPA (0.023) had a 36% and 8% reduction in FPR, respectively, when compared with FPA (0.025).

Furthermore, the detection performance of FPA, Cas_FPA, and RoF_FPA on the SMOTE-balanced Malgenome and Drebin datasets was compared. With this comparison, we aimed to ascertain whether the Cas_ FPA and RoF_FPA algorithms can be further enhanced by deploying the SMOTE data sampling method. Tables 12 and 13 present the detection performances of FPA, Cas_FPA, and RoF_FPA on the SMOTE-balanced Malgenome and Drebin datasets, respectively.

As presented in Table 12, it can be observed that Cas_FPA and RoF_FPA performed better than FPA on the SMOTE-balanced Malgenome dataset. In particular, Cas_FPA and RoF_FPA recorded +0.57% and +0.22% increments in detection accuracy values over FPA. Additionally, Cas_FPA and RoF_FPA had 45% and 18% reductions, respectively, in FPR values over FPA. In addition, a similar occurrence was observed on the SMOTE-balanced Drebin dataset. As shown in Table 13, Cas_FPA and RoF_FPA recorded +0.58% and +0.12% increments in detection accuracy values and 31.25% and 6.25% reductions, respectively, in FPR values over FPA.

Based on the analyses presented here, it can be deduced that the enhanced variants (Cas_FPA and RoF_FPA), especially Cas_FPA, are more effective in detecting Android malware than FPA. In other words, the meta-learners (cascading generalization and rotation forest) that were deployed amplified the detection performance of FPA.

For generalizability, the detection performance of FPA, Cas_FPA, RoF_FPA, Cas_FPA+SMOTE, and RoF_FPA+SMOTE were compared with that of existing state-of-the-art Android detection models. Tables 14 and 15 display the detection performance of the proposed methods and existing models on the Malgenome and Drebin datasets, respectively.

In Table 14, the results of the proposed methods are compared with findings from Lopez and Cadavid [70]; Yerima, Sezer, McWilliams, and Muttik [71]; Su, Chuah, and Tan [72] (DT); and Sen, Aysan and Clark [11]. Specifically, Lopez and Cadavid [70] developed a kNN-based Android malware model with a detection accuracy of 94% and an F-measure value of 0.940. SAFEDroid, developed by Sen, Aysan, and Clark [11], achieved a detection accuracy of 98.30% and an FPR value of 0.02. Additionally, the Bayesian-based Android malware model by Yerima, Sezer, McWilliams, and Muttik [71] had a detection accuracy of 92.10% and AUC and FPR values of 0.972 and 0.061, respectively. All these methods were trained and tested using the Malgenome dataset. Clearly, the proposed models (FPA, Cas_FPA, and RoF_FPA) showed significant improvement over the existing Android malware detection solutions.

Table 14. Detection performances of proposed methods and existing models on the Malgenome dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA	98.94	0.998	0.989	0.989	0.989	0.989	0.016
* Cas_FPA	99.45	**1**	**0.994**	**0.994**	**0.994**	**0.994**	0.008
* RoF_FPA	99.00	0.999	0.990	0.990	0.990	0.990	0.018
* Cas_FPA+SMOTE	**99.42**	**1**	**0.994**	**0.994**	**0.994**	**0.994**	**0.006**
* RoF_FPA+SMOTE	99.07	**1**	0.991	0.991	0.991	0.991	0.009
Lopez and Cadavid [70]	94.00	-	0.940	0.950	0.950	-	-
Yerima et al. [71]	92.10	0.972	-	0.937	-	0.904	0.061
Su et al. [72] (DT)	-	-	-	-	-	0.916	-
Su, Chuah and Tan [72] (RF)	-	-	-	-	-	0.967	-
Sen, Aysan and Clark [11]	98.30	-	-	-	-	-	0.020

Furthermore, as presented in Table 15, the proposed methods were compared with findings from Frenklach, Cohen, Shabtai, and Puzis [73]; Rana, Rahman, and Sung [64]; Tanmoy, Pierazzi, and Subrahmanian [74]; Salah, Shalabi, and Khedr [75]; Rana and Sung [66]; and Rathore, Sahay, Chaturvedi, and Sewak [34]. These existing Android malware detection models were trained and tested using the Drebin dataset used in the

present research. An Android malware detection model-based similarity graph proposed by Frenklach, Cohen, Shabtai, and Puzis [73] had an F-measure value of 0.869, which is lower than the F-measure values of the proposed models. Rana, Rahman, and Sung [64] developed a tree-based model with a detection accuracy of 97.92%. Additionally, Tanmoy, Pierazzi, and Subrahmanian [74] suggested an ensemble of classification and clustering (EC2) method with an F-measure value of 0.970. Salah, Shalabi, and Khedr [75] utilized a feature-selection-based framework for Android malware detection with a detection accuracy value of 94%. Similarly, the solutions produced by Rana and Sung [66], as well as Rathore, Sahay, Chaturvedi, and Sewak [34], had detection accuracy values of 97.24% and 97.92%, respectively. Although these existing methods achieved relatively good detection performances, they were still outperformed by the proposed FPA and its enhanced variants (Cas_FPA and RoF_FPA).

Table 15. Detection performances of proposed methods and existing models on the Drebin dataset.

	Accuracy	AUC	F-Measure	Precision	Recall	TPR	FPR
* FPA	98.13	0.997	0.981	0.981	0.981	0.981	0.025
* Cas_FPA	98.83	0.999	0.988	0.988	0.988	0.988	0.016
* RoF_FPA	98.38	0.997	0.984	0.984	0.984	0.984	0.023
* Cas_FPA+SMOTE	**98.94**	**0.999**	**0.989**	**0.990**	**0.989**	**0.989**	**0.011**
* RoF_FPA+SMOTE	98.49	0.998	0.985	0.985	0.985	0.985	0.015
Frenklach et al. [73]	-	-	0.869	-	-	0.939	-
Rana, Rahman and Sung [64]	97.92	-	-	-	-	-	-
Tanmoy et al. [74]	-	-	0.970	-	-	-	-
Salah et al. [75]	94.00	-	-	-	-	-	-
Rana and Sung [66]	97.24	-	0.972	0.976	-	0.969	0.239
Rathore, Sahay, Chaturvedi and Sewak [34]	97.92	-	-	-	0.976	-	-

4.3. Findings Based on Research Questions

In response to the research question raised in the introductory section, the following conclusions were drawn based on the experiments conducted.

RQ1: How effective is the FPA algorithm in comparison to baseline classifiers in Android malware detection?

It was observed from the experimental results that FPA produced significantly improved detection performance when compared with the baseline classifiers. This superior detection performance was observed on both the Malgenome and Drebin datasets.

RQ2: How effective are the enhanced variants of FPA (Cas_FPA and RoF_FPA) in Android malware detection?

According to the experimental results and analyses, Cas_FPA and RoF_FPA performed better than FPA alone on both the original and SMOTE-balanced Malgenome and Drebin datasets. Additionally, it was observed that the deployed data sampling method resolved the latent class imbalance problem and subsequently improved the detection performances of the models evaluated, especially FPA, Cas_FPA, and RoF_FPA.

RQ3: How well do the proposed FPA and its variants perform as compared to current state-of-the-art methods in Android malware detection?

It was gathered from the experimental results that the proposed FPA, Cas_FPA, and RoF_FPA algorithms, in most cases, had superior detection performance compared to existing state-of-the-art Android malware detection models.

5. Threats to Validity

Threats to validity encountered throughout the experiment are described in this section. According to Pan et al. [76], Android malware detection is becoming more relevant, and

a critical component of every empirical research is analysing and mitigating risks to the validity of experimental findings.

External validity: External validity is primarily concerned with the capacity to generalize the experimental investigation. The type and number of datasets used in the experimentation process may have an impact on the generalizability of the research findings. As such, two prominent and widely used Android malware datasets with varied numbers of malware families (Malgenome, 49; Drebin, 179) were investigated. These datasets are publicly available and are regarded as de facto Android malware datasets for training and testing Android detection models. In addition, we presented a detailed analysis of the experimentation process, which can aid in replication and validation of our experimental processes on various Android malware datasets.

Internal validity: Internal validity emphasizes the importance and consistency of datasets, ML methods, and empirical analysis. To address this issue, prominent ML methods developed and implemented in existing studies were utilized in this research work. The ML methods were chosen based on their performance and heterogeneity. Additionally, to eliminate unintended mistakes in experimental results, the investigated Android malware models were carefully deployed on the selected malware datasets using the cross-validation method, repeating each experiment 10 times for completeness. Nonetheless, future research should consider alternative model evaluation approaches and strategies.

Construct validity: Construct validity is concerned with the selection of performance measurements used to assess the efficacy of experimental Android malware detection models. In this research work, accuracy, AUC, F-measure, precision, recall, FPR, and TPR were utilized. These metrics provided an elaborate and comprehensive detection analysis of the experimented Android malware detection models. Additionally, the investigated models for Android malware detection were designed to exclusively identify whether an app is benign or malicious.

6. Conclusions and Future Work

In this study, forest penalizing attributes (FPA) and its enhanced novel variants were deployed for the detection of Android malware. Specifically, cascade generalization (Cas_FPA) and rotation-forest-based FPA (RoF_FPA) were implemented and empirically evaluated on imbalanced and SMOTE-balanced Android malware detection datasets. Experiments were carried out to investigate the effectiveness and usefulness of the proposed approaches. Results from empirical evaluation revealed that FPA produced a significantly improved detection performance when compared with the baseline classifiers, such as NB, BN, CR, DTable, ADT, and DS, on the original and SMOTE-balanced Malgenome and Drebin datasets. This observation supports the applicability of FPA for Android malware detection. Additionally, the enhanced variants of FPA (Cas_FAP and RoF_FPA) convincingly outperformed FPA on original and balanced datasets, demonstrating their effectiveness in Android malware detection. It is worth noting that the deployment of a data sampling method (SMOTE) in this research work did not only resolve the latent class imbalance problem but positively enhanced the detection performance of the proposed models. For generalizability, the proposed models (FPA, Cas_FPA, RoF_FPA, Cas_FPA+SMOTE, and RoF_FPA+SMOTE) also outperformed the state-of-the-art models in the existing literature on the Malgenome and Drebin malware datasets. Therefore, we recommend the proposed models for use in Android malware detection and the use of the data sampling method for addressing the class imbalance problem.

In the future, we plan to extend this work by introducing ensemble methods to the model and by considering hybrid (dynamic and static) features of malware applications. More real-life datasets should also be considered in future work. The high dimensionality of Android malware datasets is another issue that needs to be addressed. Specifically, 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.

Author Contributions: Conceptualization, A.G.A., A.O.B., L.F.C. and F.E.U.-H.; Data curation, A.O.B., S.A.S. and Z.O.A.; Formal analysis, A.G.A., A.O.B., S.A.S. and Z.O.A.; Funding acquisition, L.F.C. and S.B.; Investigation, A.G.A., A.O.B., H.A.M., F.E.U.-H. and G.B.B.; Methodology, A.O.B. and F.E.U.-H.; Project administration, L.F.C. and S.B.; Resources, H.A.M., S.B., S.A.S., P.O.S. and Z.O.A.; Software, H.A.M., P.O.S. and G.B.B.; Supervision, L.F.C. and S.B.; Validation, L.F.C., S.B. and P.O.S.; Visualization, S.A.S., P.O.S. and G.B.B.; Writing—original draft, A.O.B. and H.A.M.; Writing—review & editing, L.F.C., F.E.U.-H., G.B.B. and Z.O.A. All authors have read and agreed to the published version of the manuscript.

Funding: The research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Available on request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kumar, P.; Gupta, G.P.; Tripathi, R. Design of anomaly-based intrusion detection system using fog computing for IoT network. *Autom. Control Comput. Sci.* **2021**, *55*, 137–147. [CrossRef]
2. Kumar, P.; Tripathi, R.; Gupta, G.P. P2IDF: A privacy-preserving based intrusion detection framework for software-defined Internet of Things-fog (SDIoT-Fog). In Proceedings of the 2021 International Conference on Distributed Computing and Networking, Nara, Japan, 5–8 January 2021; pp. 37–42.
3. Khoda, M.E.; Kamruzzaman, J.; Gondal, I.; Imam, T.; Rahman, A. Malware detection in edge devices with fuzzy oversampling and dynamic class weighting. *Appl. Soft Comput.* **2021**, *112*, 107783. [CrossRef]
4. Alsariera, Y.A.; Balogun, A.O.; Adeyemo, V.E.; Tarawneh, O.H.; Mojeed, H.A. Intelligent Tree-based Ensemble Approaches for Phishing Website Detection. *J. Eng. Sci. Technol.* **2022**, *17*, 563–582.
5. Yerima, S.Y.; Sezer, S. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. *IEEE Trans. Cybern.* **2018**, *49*, 453–466. [CrossRef] [PubMed]
6. Alswaina, F.; Elleithy, K. Android malware family classification and analysis: Current status and future directions. *Electronics* **2020**, *9*, 942. [CrossRef]
7. Kumar, P.; Gupta, G.P.; Tripathi, R. Toward design of an intelligent cyber attack detection system using hybrid feature reduced approach for iot networks. *Arab. J. Sci. Eng.* **2021**, *46*, 3749–3778. [CrossRef]
8. McLaughlin, N.; Martinez del Rincon, J.; Kang, B.; Yerima, S.; Miller, P.; Sezer, S.; Safaei, Y.; Trickel, E.; Zhao, Z.; Doupé, A. Deep android malware detection. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 301–308.
9. Karbab, E.B.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
10. Kumar, P.; Gupta, G.P.; Tripathi, R. An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for IoMT networks. *Comput. Commun.* **2021**, *166*, 110–124. [CrossRef]
11. Sen, S.; Aysan, A.I.; Clark, J.A. SAFEDroid: Using structural features for detecting android malwares. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Niagara Falls, ON, Canada, 22–25 October 2017; pp. 255–270.
12. Kouliaridis, V.; Barmpatsalou, K.; Kambourakis, G.; Chen, S. A survey on mobile malware detection techniques. *IEICE Trans. Inf. Syst.* **2020**, *103*, 204–211. [CrossRef]
13. Kaspersky. Mobile Malware Evolution 2020. Available online: https://securelist.com/mobile-malware-evolution-2020/101029/ (accessed on 21 December 2021).
14. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A review of android malware detection approaches based on machine learning. *IEEE Access* **2020**, *8*, 124579–124607. [CrossRef]
15. Almomani, I.; Qaddoura, R.; Habib, M.; Alsoghyer, S.; Al Khayer, A.; Aljarah, I.; Faris, H. Android Ransomware Detection Based on a Hybrid Evolutionary Approach in the Context of Highly Imbalanced Data. *IEEE Access* **2021**, *9*, 57674–57691. [CrossRef]
16. Almohaini, R.; Almomani, I.; AlKhayer, A. Hybrid-Based Analysis Impact on Ransomware Detection for Android Systems. *Appl. Sci.* **2021**, *11*, 10976. [CrossRef]
17. Aslan, Ö.A.; Samet, R. A comprehensive review on malware detection approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]
18. Majid, A.-A.M.; Alshaibi, A.J.; Kostyuchenko, E.; Shelupanov, A. A review of artificial intelligence based malware detection using deep learning. *Mater. Today Proc.* **2021**. [CrossRef]
19. Mijwil, M.M. Malware Detection in Android OS using Machine Learning Techniques. *Data Sci. Appl.* **2020**, *3*, 5–9.

20. Dhalaria, M.; Gandotra, E. Android Malware Detection using Chi-Square Feature Selection and Ensemble Learning Method. In Proceedings of the 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 6–8 November 2020; pp. 36–41.
21. Dhalaria, M.; Gandotra, E. A Framework for Detection of Android Malware using Static Features. In Proceedings of the 2020 IEEE 17th India Council International Conference (INDICON), New Delhi, India, 10–13 December 2020; pp. 1–7.
22. Agrawal, P.; Trivedi, B. Machine learning classifiers for Android malware detection. In *Data Management, Analytics and Innovation*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 311–322.
23. Amouri, A.; Alaparthy, V.T.; Morgera, S.D. A machine learning based intrusion detection system for mobile Internet of Things. *Sensors* **2020**, *20*, 461. [CrossRef] [PubMed]
24. Hussain, M.S.; Khan, K.U.R. A survey of ids techniques in manets using machine-learning. In *Proceedings of the Third International Conference on Computational Intelligence and Informatics*; Springer: Singapore, 2020; pp. 743–751.
25. Alsariera, Y.A.; Adeyemo, V.E.; Balogun, A.O.; Alazzawi, A.K. Ai meta-learners and extra-trees algorithm for the detection of phishing websites. *IEEE Access* **2020**, *8*, 142532–142542. [CrossRef]
26. Balogun, A.O.; Basri, S.; Abdulkadir, S.J.; Hashim, A.S. Performance analysis of feature selection methods in software defect prediction: A search method approach. *Appl. Sci.* **2019**, *9*, 2764. [CrossRef]
27. Balogun, A.O.; Basri, S.; Abdulkadir, S.J.; Adeyemo, V.E.; Imam, A.A.; Bajeh, A.O. Software defect prediction: Analysis of class imbalance and performance stability. *J. Eng. Sci. Technol.* **2019**, *14*, 3294–3308.
28. Balogun, A.O.; Lafenwa-Balogun, F.B.; Mojeed, H.A.; Adeyemo, V.E.; Akande, O.N.; Akintola, A.G.; Bajeh, A.O.; Usman-Hamza, F.E. SMOTE-Based Homogeneous Ensemble Methods for Software Defect Prediction. In Proceedings of the International Conference on Computational Science and Its Applications, Cagliari, Italy, 1–4 July 2020; Springer: Cham, Switzerland, 2020; pp. 615–631.
29. Leevy, J.L.; Khoshgoftaar, T.M.; Bauder, R.A.; Seliya, N. A survey on addressing high-class imbalance in big data. *J. Big Data* **2018**, *5*, 1–30. [CrossRef]
30. Rokach, L. Decision forest: Twenty years of research. *Inf. Fusion* **2016**, *27*, 111–125. [CrossRef]
31. Cen, L.; Gates, C.S.; Si, L.; Li, N. A probabilistic discriminative model for android malware detection with decompiled source code. *IEEE Trans. Dependable Secur. Comput.* **2014**, *12*, 400–412. [CrossRef]
32. Fereidooni, H.; Conti, M.; Yao, D.; Sperduti, A. ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In Proceedings of the 2016 8th IFIP international conference on new technologies, mobility and security (NTMS), Larnaca, Cyprus, 21–23 November 2016; pp. 1–5.
33. Sahs, J.; Khan, L. A machine learning approach to android malware detection. In Proceedings of the 2012 European Intelligence and Security Informatics Conference, Odense, Denmark, 22–24 August 2012; pp. 141–147.
34. Rathore, H.; Sahay, S.K.; Chaturvedi, P.; Sewak, M. Android malicious application classification using clustering. In Proceedings of the International Conference on Intelligent Systems Design and Applications, Vellore, India, 6–8 December 2018; Springer: Cham, Switzerland, 2018; pp. 659–667.
35. Nisa, M.; Shah, J.H.; Kanwal, S.; Raza, M.; Khan, M.A.; Damaševičius, R.; Blažauskas, T. Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features. *Appl. Sci.* **2020**, *10*, 4966. [CrossRef]
36. Vinayakumar, R.; Soman, K.; Poornachandran, P.; Sachin Kumar, S. Detecting Android malware using long short-term memory (LSTM). *J. Intell. Fuzzy Syst.* **2018**, *34*, 1277–1288. [CrossRef]
37. Zegzhda, P.; Zegzhda, D.; Pavlenko, E.; Ignatev, G. Applying deep learning techniques for Android malware detection. In Proceedings of the 11th International Conference on Security of Information and Networks, Amalfi, Italy, 5–7 September 2018; pp. 1–8.
38. El Fiky, A.H. Deep-Droid: Deep Learning for Android Malware Detection. *Int. J. Innovative Technol. Explor. Eng.* **2020**, *9*, 122–125. [CrossRef]
39. Aslan, Ö.; Yilmaz, A.A. A New Malware Classification Framework Based on Deep Learning Algorithms. *IEEE Access* **2021**, *9*, 87936–87951. [CrossRef]
40. Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. EfficientNet Convolutional Neural Networks-based Android Malware Detection. *Comput. Secur.* **2022**, *115*, 102622. [CrossRef]
41. Gao, H.; Cheng, S.; Zhang, W. GDroid: Android malware detection and classification with graph convolutional network. *Comput. Secur.* **2021**, *106*, 102264. [CrossRef]
42. Ou, F.; Xu, J. S3Feature: A static sensitive subgraph-based feature for android malware detection. *Comput. Secur.* **2022**, *112*, 102513. [CrossRef]
43. Rahman, S.S.M.M.; Saha, S.K. StackDroid: Evaluation of a multi-level approach for detecting the malware on android using stacked generalization. In Proceedings of the International Conference on Recent Trends in Image Processing and Pattern Recognition, Solapur, India, 21–22 December 2018; Springer: Singapore, 2018; pp. 611–623.
44. Christiana, A.O.; Gyunka, B.A.; Oluwatobi, A.N. Optimizing Android Malware Detection Via Ensemble Learning. *IJIM* **2020**, *14*, 61. [CrossRef]
45. Gupta, D.; Rani, R. Improving malware detection using big data and ensemble learning. *Comput. Electr. Eng.* **2020**, *86*, 106729. [CrossRef]

46. Oak, R.; Du, M.; Yan, D.; Takawale, H.; Amit, I. Malware detection on highly imbalanced data through sequence modeling. In Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, London, UK, 15 November 2019; pp. 37–48.
47. Xu, Y.; Wu, C.; Zheng, K.; Niu, X.; Yang, Y. Fuzzy–synthetic minority oversampling technique: Oversampling based on fuzzy set theory for Android malware detection in imbalanced datasets. *Int. J. Distrib. Sens. Netw.* **2017**, *13*, 1550147717703116. [CrossRef]
48. Dehkordy, D.T.; Rasoolzadegan, A. A new machine learning-based method for android malware detection on imbalanced dataset. *Multimed. Tools Appl.* **2021**, *80*, 24533–24554. [CrossRef]
49. Adnan, M.N.; Islam, M.Z. Forest PA: Constructing a decision forest by penalizing attributes used in previous trees. *Expert Syst. Appl.* **2017**, *89*, 389–403. [CrossRef]
50. Alsariera, Y.A.; Elijah, A.V.; Balogun, A.O. Phishing website detection: Forest by penalizing attributes algorithm and its enhanced variations. *Arab. J. Sci. Eng.* **2020**, *45*, 10459–10470. [CrossRef]
51. Balogun, A.O.; Adewole, K.S.; Bajeh, A.O.; Jimoh, R.G. Cascade Generalization Based Functional Tree for Website Phishing Detection. In Proceedings of the International Conference on Advances in Cyber Security, Penang, Malaysia, 24–25 August 2021; Springer: Singapore, 2021; pp. 288–306.
52. Rodriguez, J.J.; Kuncheva, L.I.; Alonso, C.J. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 1619–1630. [CrossRef] [PubMed]
53. Tasci, E. A meta-ensemble classifier approach: Random rotation forest. *Balk. J. Electr. Comput. Eng.* **2019**, *7*, 182–187. [CrossRef]
54. Balogun, A.O.; Bajeh, A.O.; Orie, V.A.; Yusuf-Asaju, W.A. Software Defect Prediction Using Ensemble Learning: An ANP Based Evaluation Method. *FUOYEJET* **2018**, *3*, 50–55. [CrossRef]
55. Jimoh, R.; Balogun, A.; Bajeh, A.; Ajayi, S. A PROMETHEE based evaluation of software defect predictors. *JCSA* **2018**, *25*, 106–119.
56. Xu, Z.; Liu, J.; Yang, Z.; An, G.; Jia, X. The impact of feature selection on defect prediction performance: An empirical comparison. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 309–320.
57. Yu, Q.; Jiang, S.; Zhang, Y. The performance stability of defect prediction models with class imbalance: An empirical study. *IEICE Trans. Inf. Syst.* **2017**, *100*, 265–272. [CrossRef]
58. Yadav, S.; Shukla, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In Proceedings of the 2016 IEEE 6th International conference on advanced computing (IACC), Bhimavaram, India, 27–28 February 2016; pp. 78–83.
59. Arlot, S.; Lerasle, M. Choice of V for V-fold cross-validation in least-squares density estimation. *J. Mach. Learn. Res.* **2016**, *17*, 7256–7305.
60. Balogun, A.O.; Basri, S.; Jadid, S.A.; Mahamad, S.; Al-momani, M.A.; Bajeh, A.O.; Alazzawi, A.K. Search-Based Wrapper Feature Selection Methods in Software Defect Prediction: An Empirical Analysis. In *Proceedings of the Computer Science On-line Conference*; Springer: Cham, Switzerland, 2020; pp. 492–503.
61. Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA data mining software: An update. *ACM Sig. Exp.* **2009**, *11*, 10–18. [CrossRef]
62. Crawley, M.J. *The R Book*; John Wiley & Sons: New York, NY, USA, 2012.
63. Rana, M.S.; Gudla, C.; Sung, A.H. Evaluating machine learning models for Android malware detection: A comparison study. In Proceedings of the 2018 VII International Conference on Network, Communication, and Computing, Taipei City, Taiwan, 14–16 December 2018; pp. 17–21.
64. Rana, M.S.; Rahman, S.S.M.M.; Sung, A.H. Evaluation of tree-based machine learning classifiers for android malware detection. In Proceedings of the International Conference on Computational Collective Intelligence, Bristol, UK, 5–7 September 2018; Springer: Cham, Switzerland, 2018; pp. 377–385.
65. Rana, M.S.; Sung, A.H. Malware analysis on Android using supervised machine learning techniques. *Int. J. Comput. Commun. Eng.* **2018**, *7*, 178. [CrossRef]
66. Rana, M.S.; Sung, A.H. Evaluation of Advanced Ensemble Learning Techniques for Android Malware Detection. *Vietnam J. Comput. Sci.* **2020**, *7*, 145–159. [CrossRef]
67. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 23–26 February 2014; ACM. pp. 1–12.
68. Zhou, Y.; Jiang, X. Dissecting android malware: Characterization and evolution. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 20–23 May 2012; pp. 95–109.
69. Aljerf, L.; Alhaffar, I. Salivary distinctiveness and modifications in males with diabetes and Behçet's disease. *Biochem. Res. Int.* **2017**, *2017*, 9596202. [CrossRef]
70. Lopez, C.C.U.; Cadavid, A.N. Machine learning classifiers for android malware analysis. In Proceedings of the 2016 IEEE Colombian Conference on Communications and Computing (COLCOM), Cartagena, Colombia, 27–29 April 2016; pp. 1–6.
71. Yerima, S.Y.; Sezer, S.; McWilliams, G.; Muttik, I. A new android malware detection approach using bayesian classification. In Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona, Spain, 25–28 March 2013; pp. 121–128.

72. Su, X.; Chuah, M.; Tan, G. Smartphone dual defense protection framework: Detecting malicious applications in android markets. In Proceedings of the 2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN), Chengdu, China, 14–16 December 2012; pp. 153–160.
73. Frenklach, T.; Cohen, D.; Shabtai, A.; Puzis, R. Android malware detection via an app similarity graph. *Comput. Secur.* **2021**, *109*, 102386. [CrossRef]
74. Tanmoy, C.; Pierazzi, F.; Subrahmanian, V. EC2: Ensemble Clustering & Classification for predicting Android malware families. *IEEE Trans. Dependable Secure Comput.* **2020**, *17*, 262–277. [CrossRef]
75. Salah, A.; Shalabi, E.; Khedr, W. A lightweight android malware classifier using novel feature selection methods. *Symmetry* **2020**, *12*, 858. [CrossRef]
76. Pan, Y.; Ge, X.; Fang, C.; Fan, Y. A systematic literature review of android malware detection using static analysis. *IEEE Access* **2020**, *8*, 116363–116379. [CrossRef]