Hindawi
Mobile Information Systems
Volume 2022, Article ID 6425583, 15 pages
https://doi.org/10.1155/2022/6425583

Research Article

CFSBFDroid: Android Malware Detection Using CFS + Best First Search-Based Feature Selection

Ravi Mohan Sharma,[1] **Chaitanya. P Agrawal**,[1] **Vinod Kumar**,[2] **and Adugna Necho Mulatu**[3]

[1]*Computer Science and Applications, Makhanlal Chaturvedi University, Bhopal, M.P 462001, India*
[2]*Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India*
[3]*Faculty of Electrical and Computer Engineering, Bahir Dar Institute of Technology, Bahir Dar University, Bahir Dar, Ethiopia*

Correspondence should be addressed to Adugna Necho Mulatu; adugna.necho@bdu.edu.et

Received 1 May 2022; Accepted 9 June 2022; Published 7 July 2022

Academic Editor: M. Praveen Kumar Reddy

With the fast development of smartphone technology and mobile applications, the mobile phone has become the most powerful tool to access the Internet and get various services with one click. Meanwhile, susceptibilities of the application are the primary hazard to the security of Android devices. Due to these weaknesses, an attacker can easily hack the confidential data of the mobile phone. The malware application automatically performs fraudulent activities on mobile phones without the user's knowledge. Thus, these attacks are the major threats to the security of mobile phones. To detect malicious applications installed on Android smartphones, we have conducted a study that focuses on permissions and intent-based mechanisms. The study was done in three phases: in the first phase, the dataset was created by extracting intents and permissions from APK files; in the second phase, correlation-based feature selection (CFS) and best first search (BFS) were combined to select the most representative features from the feature space of the extracted dataset; and in the third phase, machine learning (ML) techniques were trained and tested against the preprocessed dataset obtained in the second phase. The accuracy, precision, recall, F1 score, and error metrics of seven machine learning techniques (REPTree, Rule PART, RF, SMO, SGD, MCC, and LMT) were demonstrated over the Android dataset.

1. Introduction

Nowadays, mobile devices have become an essential part of people's lives. According to Statista [1], the number of smartphone users worldwide is around 6.4 billion and is likely to increase several hundred million in the next few years. The Google Play Store is the largest app store in the world, and it will have 3.48 million apps by the first quarter of 2021. Most daily activities such as online shopping, bill payment, and mobile banking are done through mobile applications. This has increased the risk of theft of confidential information such as bank details, credit and debit cards, and ATM PINs. Cybercriminals always remain active in attacks on various mobile payment systems, including brute force attacks to obtain user's PINs, account manipulation, and redirecting fake traffic to mobile money servers [2]. Future mobile is compatible with many applications, including AI-powered health care, mobile edge computing, and innovative industry applications; therefore, future smartphones must be equipped with the latest high-level security mechanisms [3].

Every app acquires a list of permission from the users at the installation time. Appropriate permissions allow a user to infer the behavior of any application. Identifying key permissions for functionalities and expected permission requests helps leverage unusual application behavior and provides a simple risk warning for users. In this way, the permission-based mechanism warns the user before installation. It gives the option for the user to understand the risk of allowing the application's permissions on the user's

mobile. For example, if an app is allowed to use the Internet, then that app can access the Internet through your mobile and consume your daily data. But due to the lack of knowledge, users cannot understand and recognize the permissions required by each application and often ignore pre-installation warnings, so this mechanism does not succeed in protecting the user from malware. According to Statista, 482,579 new malwares are added monthly. Therefore, it is necessary to detect malware on mobile devices. If we can detect malware during installation, we can stop the malicious app installation. We need a scalable malware detection approach to combat this severe malware attack, which can effectively and efficiently detect malware applications. However, scaling the detection for a large number of apps is a challenging task. Finding a way to identify malware is a big problem that needs to resolve immediately. This paper introduces a static analysis-based malware detection system to protect against Android malware. This research presents a novel and unique contribution, which are as follows:

(i) This research creates an intent and permission-based feature set using the APK and Aapt2 tools.

(ii) A new framework for identifying the optimal subset combines the CFS and Best first search techniques.

(iii) A total of 3000 app samples were evaluated using seven machine learning (ML) techniques.

(iv) The proposed method can detect malware in real-world apps in a shorter amount of time.

The remaining sections of the paper are arranged as follows: Section 2 goes over previous works on Android malware detection and their limitations. Section 3 describes the different machine learning techniques applied. Section 4 contains the proposed methodology. Section 5 describes the experimental arrangement and various performance evaluation matrices. Section 6 presents the obtained results. Section 7 concludes the works, followed by references.

2. Related Works

The explosive growth of Android malware, well as its destructive nature, motivates researchers to conduct malware analysis. Malware analysis is classified into two types: signature-based analysis and behavior-based analysis. To identify malware, the signature-based detection method compares the binary code of an app with the binary code of known malware. As a result, this method necessitates massive databases of known malware. Hence, signature-based techniques are simple, efficient, and accurate, but they cannot identify unknown malware [4].

On the other hand, behavior-based detection compares the behavior patterns of applications with the behavior patterns of known malware [5]. The behavior-based method is commonly used to detect unknown malware. But this method yields a high false alarm rate. Typically, malware analysis is done in three ways, static, dynamic, and hybrid. The static analysis tested the application without actually running it on a real environment. In the dynamic approach, the application is tested by running it in a specific environment [6]. And hybrid approach combines the characteristics of static and dynamic methods. This section will review some of the efforts made in three categories.

2.1. Static Analysis. Lou et al. [7] developed the TFDroid method to detect malware applications. They used topics and sensitive data flows as a feature set. Using machine learning (ML) techniques, they achieve a classification accuracy of 93.7%. However, this approach suffers from a high false alarm rate. In another paper, Taheri et al. [8] proposed the hamming distance-based approach to computing a similarity scores between malware and benign ware. They used three datasets for the experimental purpose and confirmed the accuracy of 90% using API call features. This method also suffers from a high false alarm rate. Wu et al. [9] presented the MVIIDroid framework. They trained multiple kernel learning (MKL) classifiers and compared the performance with other methods such as RF, JS, and SVM, and confirmed an accuracy of 94.8% for Android family classification and 99% for simple malware detection. Feng et al. [10] proposed a two-layer deep learning (DL) model where the first layer analyzes the static features. The second layer inspects network flow data for malware detection and performs 99.3% accuracy. Li et al. [11] proposed the SIGPID malware detection approach, which selects significant features from the dataset and performs 93.62 percent accuracy using SVM classifiers. This method suffers from high computational complexity. Taheri et al. [12] created the "CICAndMal-2017" dataset using the network flow and API call features. This approach achieved 95.3% precision in static classification at the first layer and 83.3% in dynamic classification. Santosh et al. [13] proposed a gain ratio method to select relevant features from the dataset and confirmed 94.2% accuracy by applying ML techniques such as J48, RC, MLP, SMO, and randomizable filters. In the paper [14], Alzaylaee and others have proposed DL-Droid, a deep learning system for detecting malicious Android applications through dynamic analysis using stateful input generation. The study shows that the detection rate of DL-Droid with dynamic features is 97.8%, and the detection accuracy with dynamic and static attributes is 99.6%, which improves the traditional machine learning technique. Chen et al. [15]presented the android malware identification by using traffic features by using three supervised machine learning methods and confirmed 95% average accuracy. Jiang et al. [16] proposed fine-grained dangerous permission (FDP) method that collects the difference between malicious apps and benign apps, performs classification, and achieves a 94.5% TP rate.

2.2. Dynamic Analysis. Dynamic analysis typically encompasses executing and testing applications in a safe environment and providing the necessary resources to identify malicious activities. Thangaveloo et al. [17] presented a dynamic DATDroid approach with 91.7 percent accuracy, 93.1 percent recall rate, and 90.0 percent accuracy. Ananya. et al. [6] presented a "sysDroid" system call trace-based dynamic approach that uses LR, CART, RF, XGBoost, and

DNN-based evaluations to achieve 95 to 99 percent accuracy. Casolare et al. [6] demonstrated a trace-based dynamic analysis system with 89 percent accuracy. Y Yang et al. [18] presented a dynamic approach named DroidWard using SVM, RF, and DT, which attained an accuracy of 98.49%, recall rate of 98.54%, and false-positive rate of 1.55%.

2.3. Hybrid Analysis. Some authors offered a hybrid method for malware detection. Amin et al. [19] demonstrated an AndroShield hybrid approach. Static analysis inspects the code without running the application and dynamic examination of the application by running the application. Zhang et al. [20] introduced a DAMBA, a hybrid malware detection method using the ORGB analysis method. They extract static and dynamic attributes from apps and propose a TANMAD malware finding algorithm and confirm the very high accuracy. Ahmed et al. [21] have designed a hybrid StaDART malware detection method for dynamic code update features. In another form [22], Gajrani et al. have presented a hybrid analysis approach EspyDroid + that incorporates a reflection-guided static slicing (RGSS) method, which helps handle C&C-controlled execution, logic bombs, time bombs, etc. In another study, Ali-Gombe et al. [23] designed the hybrid analysis technique named AspectDroid in which 100 malware and 100 benign ware apps were evaluated and achieved 94.68% accuracy. Some other essential works and their limitation are summarized in Table 1.

From the brief literature review discussed above and the literature in Table 1, it is clear that the previous research had the following limitations: an imbalanced dataset, a long detection time, and a high FPR. To overcome the above problem, we created the balanced dataset with 1500 malware and 1500 benign ware samples. Furthermore, to overcome the computational complexity and reduce the detection time, we implemented a lightweight approach, which combine the CFS and Best First search, which provides distinctive features that will help to reduce computation time and FPR.

3. Description of Employed ML Techniques

This section describes the seven ML techniques employed to detect malware using the hypothesis that the developed model has minimum detection time, higher accuracy, and lower error rate.

3.1. Random Forest (RF) Classifier. It is a classifier that uses a number of decision trees on different subsets of a given dataset and averages them to improve the predictive accuracy of that dataset. For example, if in a random forest method has six decision trees and three classes, namely, A, B, and C. If three of these trees predict class A, then class A's score will be three, and if two trees predict class B, then class B's score will be two, and class C's score is one. Thus, class "A" has the highest score among the three classes. So, class "A" will be the predicted class. The random forest method determines the relative importance of each attribute, reducing the variance and reducing the possibility of overfitting. It also reduces computational cost and training time [39] Algorithm 1 shows the Pseudocode of Random Forest(RF) classifier.

3.2. Reduced Error Pruning Tree (REPTree) Classifier. REPTree is a fast and straightforward decision/regression tree-based classification approach. It uses the information gain/variance value, and the prunes are used for obtaining a reduced-error pruning tree (with back-fitting). The information gain value is used to create a node in the decision tree. Let D denote the training dataset, which $D = \{X1, X2, X3, \ldots, Xn, Y\}$, where X is the attributes, and Y is the class label [39] Algorithm 2 shows the Pseudocode of REPTree Classifier.

3.3. Rule PART Classifier. The Rule PART method applies a divide-and-conquer strategy with a separate-and-conquer strategy for classification. The quality of classification is dependent on the coverage function. A simple pseudocode of the Rule PART method is given below Algorithm 3.

3.4. Logistic Model Tree (LMT) Classifier. LMT is a classification tree that deals with binary and multiclass classification. It applies a logistic regression function at the leaves and works with numeric and nominal values. The LMT uses cross-validation to find LogitBoost iterations that do not overfit the training data [40].

3.5. Sequential Minimal Optimization (SMO) Support Vector Classifier (Linear Kernel). The SMO is a method of decomposition in which the problem of multiple variables is decomposed into a series of subproblems, which optimizes an objective function. It usually takes one variable at a time, and other variables are treated as constant and remain unchanged. An SMO solves SVM-QP (Support Vector Machine Quadratic Programming) problems by breaking them down into the smallest possible subproblems and incorporating two Lagrange multipliers at each step. These minor quadratic programming issues are solved analytically, avoiding using a time-consuming numerical quadratic programming optimization as an inner loop. An SMO can handle a large dataset and optimizes Lagrange multipliers using a heuristic. It rapidly solves the SVM-QP problem without the need for additional matrix storage [41].

3.6. Meta Multiclass Classifier (MCC). In this classification, meta multiclass classifier uses a one-agents-all heuristic method, which divides the multiclass dataset into multiple binary classification problems. It trains a binary classifier and predicts the model that produces the highest confidence score [42]. The pseudocode for the multiclass classifier is given as follows Algorithm 4.

Since the dataset used in this study only has binary labels, the multiclass classifier behaves like a binary classifier.

3.7. Stochastic Gradient Descent (SGD) Classifier. The term "stochastic" refers to a process associated with a random probability, and gradient descent is a popular

Table 1: Summarized related works from the literature.

S. No.	Reference	Analysis type	Dataset	Features	Applied techniques	Performance claimed	Limitation
1.	Zhu et al., 2021 [24]	Static	1065 (B) 1065 (M)	Sensitive API monitoring system event	Ensemble rotation forest	Accuracy—88.26%	The variation between MCC and accuracy
2	Firdaus et al., 2018 [25]	Static	550 (B) 5555 (M)	Permission rateCodebase feature string, permission directory path, etc.	NB, FT, J48 RF, and MLP	Accuracy—95%	High FPR and imbalance dataset
3	Martinelli et al., 2020 [26]	Hybrid	9804 (B) 2794 (M)	Static-n-grams dynamic monitoring devices, apps behavior, etc.	SVM	Accuracy—99.7%	Only one classifier is used for evaluation and the imbalance dataset
4	S. Alam et al., 2020 [27]	Dynamic	500 (B) 200 (M)	Network traffic	J48	Accuracy—98.4%	Only one classifier is used for evaluation and high FPR
5	Sugunan et al., 2018 [28]	Hybrid	200 (B) 150 (M)	Permission,API calls	NB, SVM, RF, andJ48	Precision—90.5%	Small sample size, variation in precision, and recall and F score
6	Feng et al., 2018 [10]	Dynamic	8806 (B) 5213 (M) 5000 (B) 5000 (M)	System calls, phone calls, and sent SMS	Majority voting stacking	Accuracy—96.56%	High FPR in system call sample
7	Martín et al., 2018 [29]	Dynamic	4442 (B + M)	System calls, SMS sent, cryptographic operation, etc.	Bagging, DT, NN, CNN LSTM, RNN, SVM linear, SVM rbf, SVM sigmoid, etc.	Accuracy—81.8%	Performance of SVM are lowest
8	Yerima et al., 2019 [14]	Dynamic	17444 (B + M)	API calls and intent	RF, MLP, SMO J48, PART, and NB	Accuracy—94.3%	Complex procedure
9	Yang et al., 2018 [30]	Dynamic	408 (B) 258 (M)	Packet size, sensitive API, antisimulator, etc.	SVM, RF, and DT	Accuracy—98.54%	Imbalance sample size
10	Surendran et al., 2020 [31]	Hybrid	1650 (B) ! 650 (M)	API calls, permissions, and system calls	TANB	Accuracy—97%	Variation in TPR and precision
11	Wang et al., 2020 [32]	Static	61436 (B) 27500 (M)	URL and HTTP traffic	MultiView SVM, NB KNN, and C4.5	Accuracy—98.8%	FPR and errors not estimated
12	Fang et al., 2020 [33]	Static	AMD	Dex files into RGB image	KNN, SVM RF, and familial classification	F1 score—96%	A small number of features considered
13	Tao et al., 2017 [34]	Hybrid	123453 (B) 5560 (M)	Permission, restricted APIs, suspicious API, network address, etc.	SVM DREBIN	Accuracy—94%	The variation in precision and recall values, and imbalance dataset
14	Garg and Baliyan, 2019 [35]	Hybrid	85000 (M + B)	Permissions, API calls, services, etc.	MLP, SVM PART, RINDOR, MaxProb, etc.	Accuracy—98.27%	High FPR and imbalance dataset
15	Maryam et al., 2020 [36]	Hybrid	2500 (B) 2500 (M)	Dex class, hashes, Fda access, permissions, etc.	SVM, DT, RF K-star, NB TPOT, etc.	F score—97%	Variation in precision and recall values
16	Jiang et al., 2020 [16]	Hybrid	4002 (B) 1886 (M)	Permission, APIs, intent filters, suspicious calls, system calls, etc.	DNN, RBM, DAE, SVM MKL etc.	Accuracy—94.7%	High false negative and false positive
17	Duc et al., 2018 [37]	Static	123453 (B) 5560 (M)	Requested permission, intent filter, API request, etc.	Neural network	Accuracy—92.3%	Variation in precision and recall values
18	Arshad et al., 2018 [34]	Hybrid	100 (B) 100 (M)	Permission, system calls, etc.	RF, DT, SVM, NB, and SAMADroid	F score—98%	Small size sample
19	Alazab et al., 2020 [38]	Static	14172 (B) 13719 (M)	API calls	RF, J48, RT KNN, and NB	F score—94.30%	FPR not estimated

Input (RF) : A training set (RF)
(1) It randomly selects the "k" attributes from the dataset.
(2) It uses the Gini index for the best split, selects the root node, and forms multiple decision trees.
(3) The forecast is made based on the outcome of these decision trees.
(4) Finally, calculate the number of votes for each class label. The highest voted class becomes the predicted class.
Output: Classified Instances

Algorithm 1: Pseudocode of random forest (RF).

Input (REPTree): A training set
(1) Build_REPtree (D, X split)
(2) {Calculate Information Gain (IG) value for each attribute X, IG (D, X) If the attribute is numeric (Find the split point) Xmax is a split attribute where the value of IG (D, X) is the maximum among all the attributes
(3) If IG (T, Xmax) > IG (T, Xsplit) {For all v ∈ Val (Xmax) {$D = \{X \in D | \text{Xmax} = v\}$ Build REPTree(D, Xmax)(vii) } } }
Output: Classified Instances

Algorithm 2: Pseudocode for REPTree classifier.

Input (Rule Part): A training set
(1) Create a partial decision tree on the present set of instances
(2) By using a DT, it creates rules for the terminal node (leaf) with the highest Coverage is made as a rule. The fraction of instances that satisfy the condition of a rule is known as the coverage function. And the situation is a combination of attribute tests.($K1 = C1$) and ($K2 = C2$) and . . . and ($Kn = Cn$) where C is the prediction class and K represents the attributes, and n is the number of records in the dataset.
(3) Discard the decision tree
(4) Eliminate the instances covered by the rule
(5) Go to step one
Output: Classified Instances

Algorithm 3: Pseudocode of the rule PART.

Input (MCC): A training set
(1) Dataset X as defined above and Prediction class (C) where $l \in \{0, 1\}$ is the class labels in dataset X
(2) For each class label l
(3) Make a new class label vector M where $M = l$ if l = (0 or 1) otherwise $M = 0$
(4) Apply binary classification B to datasets X and M to obtain f(l). f(l) is a list of binary classifiers All the classifiers applied to dataset X, and those whose confidence score is high are selected.
(5) The confidence score can be defined as follows confidence score = arg max f(l) (x) All the classifiers applied to dataset X, and those whose confidence score is high are selected.
Output: Classified Instances

Algorithm 4: Pseudocode for the multiclass classifier.

optimization method used in deep learning and machine learning. The gradient descent algorithm finds the best possible values for the parameters of a given cost function. For each iteration of stochastic gradient descent, some samples are chosen at random rather than the entire dataset. The term "batch" in gradient descent refers to the total number of samples from a dataset that are used to calculate gradients for each iteration. The gradient is a function's slope that measures the degree of change of one variable with respect to the changes in another variable. Gradient descent is a convex function whose output is a partial derivative of the input parameters. It reduces the computational load, particularly in high-dimensional optimization problems, allowing faster iterations for lower convergence rates. The stochastic gradient descent (SGD) classifier works as follows Algorithm 5.

Input: Training dataset

(1) If "S" is the size of the training dataset and $f(x)$ is the loss function on the data instance index by "k." $f(x) = 1/s \sum_{k=0}^{s} f_k(x)$

(2) If the size of the dataset is huge, then the gradient descent function may be infeasible due to the high computational cost. For the large dataset, the stochastic gradient descent (SGD) provides a lighter-weight solution for each iteration; instead of calculating the gradient ∇f(x), the SGD randomly selects the "k" sample from the dataset and calculates $\nabla fk(X)$ as an unbiased estimator of $\nabla f(x).\nabla f(X) = 1/s \sum_{k=0}^{s} \nabla f_k(X)$

(3) At each iteration, a mini-batch μ $(\nabla f(x) = 1/|\mu| \sum_k \int_\mu \nabla f_k(x))$ to update X as $x = x - \eta \nabla f(X)$ {where η {\displaystyle\eta} is a step size}, where $|\mu|$ is a mini-batch size and is a positive scalar, which represents the learning rate or steps size. This generalized stochastic algorithm is also called mini-batch SGD. The computational cost of per-iteration is O $(|\mu|)$. Thus, when the mini-batch size is small, the computational cost is light at each iteration.

Output: Classified Instances

Algorithm 5

4. Proposed Methodology

This section explains the overall methodology that has been proposed. This section is divided into three sections: the first section describes the proposed architecture, the second section deals with feature extraction and dataset preparation, and the third section explains how to choose the best features using the CFS and Best First search (BFS) technique.

4.1. Proposed Architecture. The overall architecture of the malware detection process is depicted in Figure 1. The six-step method is included in this architecture. The first step deals with the collection of benign and malicious APK (Android Package Kit) files from "CICAndMal2017." The second phase involves using APKTOOL to decompose APK files and the AAPT2 tool to extract features. The dataset is created in CSV format in the third stage, and data preparation is done in the fourth step. Classification is performed in the fifth stage using a variety of machine learning algorithms. The outcomes are reviewed using different performance evaluation metrics in the final stage.

4.2. Feature Extraction and Dataset Creation. This section describes the feature extraction and dataset preparation process. The Android Package Kit (APK) is the file format used by the Android operating system to distribute and install apps on Android devices. An APK package contains everything needed for an application to be correctly installed and operate on a mobile device. Malware and benign files are downloaded from "CICAndmal2017" [43] and use the APK TOOL [44] to decompile APK files to obtain the necessary information. The AAPT2 tool [45] is used for extracting permissions and intents from the AndroidManifest.xml. The process of feature extraction and dataset creation is illustrated in algorithm 1 and Figure 2.

4.3. Optimal Subset Selection. The combination of correlation-based feature selection (CFS) and Best First techniques is used to assess the optimality of features from the dataset. Combining both methods selects 78 out of 300 attributes from the dataset.

4.3.1. CFS Subset Evaluation. Correlation-based feature selection assesses the value of a subset of attributes by considering each feature's predictive ability and the degree of redundancy between them.

4.3.2. Best First. Best First searches the space of attribute subsets using greedy Hill-climbing with a backtracking facility supplements. The amount of backtracking done is controlled by the number of consecutive nonimproving nodes allowed. It is best to begin with an empty set of attributes and search onward, begin with a complete set of features and search backward, or begin at any point and search in both directions (by considering all possible single attribute additions and deletions at a given moment) Algorithm 6 shows the steps of Creating Feature Set (Dataset).

5. Experimental Environments and Performance Evaluation Matrix

The datasets produced by the previous process have been preprocessed. All entries are reviewed during this phase, and data cleaning and filtering are performed. After effectively preprocessing the dataset, the optimal subset was obtained using the CFS + Best First method. Following that, 70% of the dataset is used for training, 30% is used for testing purposes, and 10-fold cross-validation is used to validate the given model. The Waikato Environment Knowledge Analysis (WEKA) tools were used in all experiments. All tests are run on a machine with 8 GB of memory and a 1.80 GHz Intel (R) Core (TM) i-7 8550U processor.

5.1. Performance Evaluation Matrix. The confusion matrix is an $N * N$ matrix used to evaluate the performance of ML techniques. Performance. The confusion matrix provides more insight into the predictive model's performance and describes which classes are classified correctly and are not.

(1) A true positive (TP): a true positive is correctly classified trials that belong to a positive class.

(2) A false positive (FP): a false positive is incorrectly classified trials that belong to a positive class.

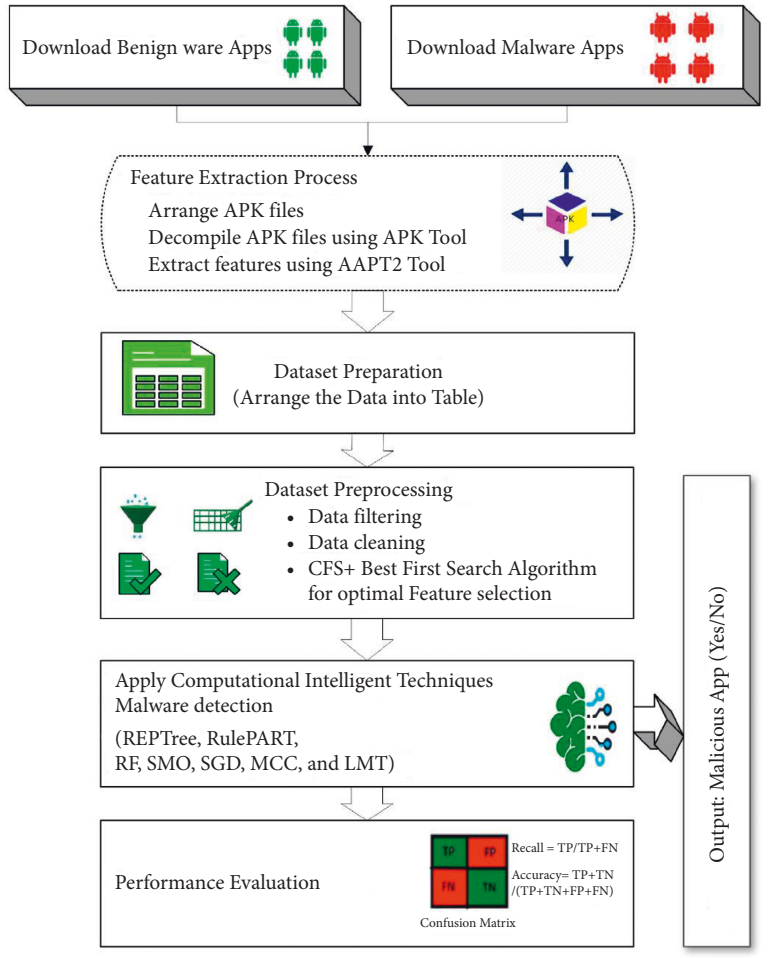


Figure 1: Architecture for malware detection in Android Device.

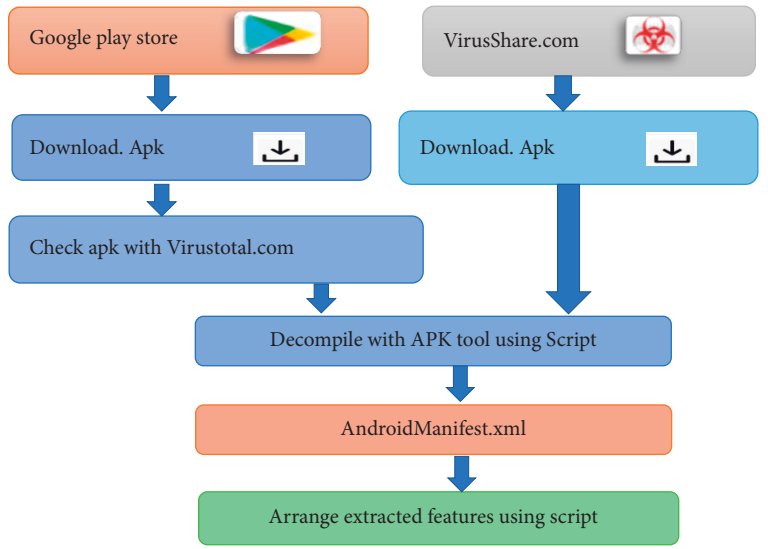


Figure 2: Feature extraction process from Android applications.

```
Input Download Benign Ware and Malware's APK from CICAndMal 2017
Output Two dimensional Dataset (Row represent the Apps and columns denotes the features)
(1) Decompile all the APK and get Androidmanifest.XML
(2) Function Create_Feature_Dataset[apps[], Permission_Intents[]]
(3) Feature_Dataset[[],[]] ¢ for empty dataset
(4) N=Count(Applications)
(5) For i = 1 to N do
(6)    Call AAPT2
(7)    Extract permissions and Intents from each Aandriodmanifest.XML file
(8)    K= Count(Permission_Intents)
(9)    For j = 1 to K do
(10)   If Permission_Intents[j] is a member of Aandriodmanifest.XML[j]
(11)   Then
(12)      Feature_Dataset[i][j] = 1
(13)   Else
(14)      Feature_Dataset[i][j] = 0
(15)   end if
(16)   end for
(17) end for
(18) return Feature_Dataset
(19) End function
```

Algorithm 6: Creating Feature Set (Dataset).

(3) A true negative (TN): a true negative is suitably classified trials that belong to a negative class.

(4) A false negative (FN): a false negative is wrongly classified trials that belong to a negative class.

We calculate the following metrics to evaluate the effectiveness of the proposed method.

Accuracy: the classification accuracy represents the classifier's performance. The accuracy is calculated using the following equation:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}. \tag{1}$$

Recall: recall is also known as sensitivity (SN) and true positive rate (TPR). The recall is a ratio of the total number of predictions that is relevant to the total number of relevant predictions. The recall is calculated using the following equation:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2}$$

Precision: the precision ratio of true-positive predictions to the total number of positive predictions (TP + FP). The precision is represented as following equation

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3}$$

F-measure: the F-measure is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy metric. The F-measure uses harmonic mean because it penalizes the extreme values. The F-measure is calculated using the following equation:

$$F - \text{measure} = \frac{2 * \text{Precision} * \text{Recall}}{(\text{Precision} + \text{Recall})}. \tag{4}$$

False positive rate (FPR) (also known as false alarm rate): the F-measure employs the harmonic mean of precision and recall. It penalizes extreme values and provides a more accurate measure of incorrectly classified cases than the accuracy metric. The false-positive rate is calculated using the following equation:

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

AUC: the AUC is an acronym for "area under the ROC curve." It is a metric that measures performance across all possible classification thresholds. The AUC is simply the area between that curve and the x-axis. The area under the ROC curve is measured using the following equation:

$$\text{AUC} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP} + \text{FP}} + \frac{\text{TN}}{\text{TN} + \text{FP}}\right). \tag{6}$$

ROC: a ROC curve (receiver operating characteristic curve) is a graph formed by plotting the true-positive rate (TPR) against the false-positive rate (FPR) at various threshold values to represent the performance of a classification model across all classifications.

MCC: the Matthews correlation coefficient (MCC) is used to measure the dominance of binary classifications. Its value lies between -1 and +1, where +1 represents the perfect classification, and −1 represents total failure to classify. The Matthews correlation coefficient is calculated using the following equation:

$$\text{MCC} = \frac{\text{TP} * \text{TN} - \text{FP} * \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}. \tag{7}$$

Table 2: Attained confusion matrix of random forest.

Class	Predicted		
	0	1	∑
0	1481	19	1500
1	48	1452	1500
∑	1529	1471	3000

Learning parameters for random forest: Bag_Size_Percent = 100, Batch_Size = 100 Max_Depth = 0, No_of_execution_slot = 0, Number of iterations = 100, seed = 1, and Number of decimal Place = 2.

Table 3: Achieved confusion matrix for REPTree.

Class	Predicted		
	0	1	∑
0	1489	11	1500
1	94	1406	1500
∑	1583	1417	3000

Learning Parameters for REPTree: Batch_Size = 100, Max_Depth = −1, Minimum variation probability = 0.001 Number of folds = 3, Seed = 1 Number of decimal Place = 2.

Table 4: Attained confusion matrix of Rule PART.

Class	Predicted		
	0	1	∑
0	1480	20	1500
1	80	1420	1500
∑	1560	1440	3000

Learning parameters for Rule PART: Batch Size = 100, Confidence_Factor = 0.25, Minimum Number of Object = 2 Number of decimal place = 2, and Seed = 1 Use MDL correction = 1.

Table 5: Obtained confusion matrix of logistic model tree (LMT).

Class	Predicted		
	0	1	∑
0	1498	2	1500
1	0	1500	1500
∑	1498	1502	3000

Learning parameters for LMT : Batch_size = 100, Minimum_No._of_ Instance = 15, Number_of_Boosting_iterations = 1, and Weight Trim $\beta = 0$.

Table 6: Obtained confusion matrix of sequential minimal optimization (SMO).

Class	Predicted		
	0	1	∑
0	1494	6	1500
1	59	1441	1500
∑	1553	1447	3000

Learning parameters for SMO : Batch_size = 100, € = 1.0E-12, $C = 1.0$, Number_of_decimal_places = 2, Number of folds = -1, Random Seed = 1, and Tolerance Parameters = 0.001.

Mean absolute error (MAE): the MAE is used to measure the prediction error in the classification problem. The absolute difference ignores the negative value. It is not very sensitive to outliers. The MAE goes from 0 to infinite the values near 0 represent the best performance. The MAE is used to calculate performance on continuous data. It gives a linear value, which averages the weighted individual differences equally. The mean absolute error is calculated using the following equation:

Table 7: Obtained confusion matrix of multiclass classifier.

Class	Predicted		
	0	1	∑
0	1498	2	1500
1	2	1498	1500
∑	1500	1500	3000

Learning parameters for MCC : Batch size = 100, Method = 1 –Against -All, Number_of_decimal_places = 2 Number of Width factor = 2, Seed = 1, and Use pair for Coupling = F.

Table 8: Obtained confusion matrix of SGD.

Class	Predicted		
	0	1	∑
0	1495	5	1500
1	27	1473	1500
∑	1522	1478	3000

Learning parameters for SGD : Batch _size = 100, Number_of_decimal_places = 2, Loss_Function = SVM, $\lambda = 1.0\text{E-}4$ $\varepsilon = 0.001$, epochs = 500, Seed = 1, and Learning rate = 0.01.

$$\text{MAE} = \frac{1}{n}\sum_{i=0}^{n}\left|\theta_p - \theta_t\right|, \tag{8}$$

where θ_t = true value, θ_p = predicted value (aspected value), and n = total instance.

Root mean squared error (RMSE): the RMSE measures are the standard deviation of the residual errors. The errors are measured by subtracting the actual value from predicted values, and the errors are squared before they are averaged. The values near 0 indicate the better performance of the model. The RMSE is very sensitive to outliers, and significant errors are penalized. The RMSE is very useful when significant errors are present and considerably influence the model's performance. The root mean squared error is calculated using the following equation:

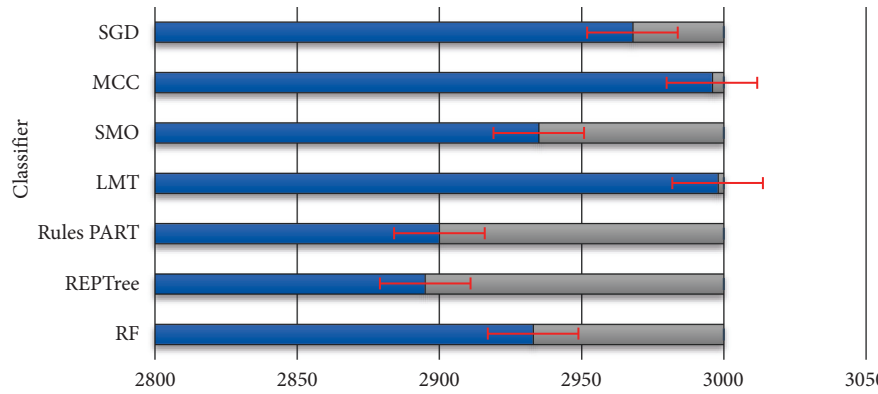
$$\text{RMSE} = \sqrt{1/n\sum_{i=0}^{n}\left(\theta_p - \theta_t\right)^2}, \tag{9}$$

where θ_t = true value, θ_p = predicted value (aspected value), and n = total instance.

Relative absolute error (RAE): the relative absolute error (RAE) compared a mean error to errors produced by a naive model and expressed as a ratio. It indicates a reasonable model (which gives a better result). It is relative because the mean difference is divided by the arithmetic mean. The value of RAE is closer to 0 represents better performance. The RAE is expressed in percentage; the formula of RAE is given below in the following equation:

$$\text{RAE} = \sum_{i=0}^{n}\left|\theta_p - \theta_t\right| / \left|\overline{\theta} - \theta_t\right|, \tag{10}$$

where θ_t = true value, θ_p = predicted value (aspected value), and n = total instance, and $\overline{\theta}$ is a mean value of θ.



	RF	REPTree	Rules PART	LMT	SMO	MCC	SGD
Correctly Clsssified instances	2933	2895	2900	2998	2935	2996	2968
Incorrectly Clsssified instances	67	105	100	2	65	4	32

No. of instances

Figure 3: Comparison of correctly and incorrectly classified instances.

Table 9: Performance of various classifiers.

Classifier	Class	TP rate	FP rate	Precision	Recall	MCC	ROC area	F-measure
RF	0	0.987	0.032	0.969	0.987	0.956	0.995	0.978
	1	0.968	0.013	0.987	0.968	0.956	0.995	0.977
	Weighted Avg.	0.978	0.022	0.978	0.978	0.956	0.995	0.978
REPTree	0	0.993	0.063	0.941	0.993	0.931	0.981	0.966
	1	0.937	0.007	0.992	0.937	0.931	0.981	0.964
	Weighted Avg.	0.965	0.035	0.966	0.965	0.931	0.981	0.965
Rule PART	0	0.987	0.053	0.949	0.987	0.934	0.980	0.967
	1	0.947	0.013	0.986	0.947	0.980	0.983	0.966
	Weighted Avg.	0.967	0.033	0.967	0.967	0.934	0.980	0.967
LMT	0	0.999	0.000	1.000	0.999	0.999	1.000	0.999
	1	1.000	0.001	0.999	1.000	0.999	1.000	0.999
	Weighted Avg.	0.999	0.001	0.999	0.999	0.999	1.000	0.999
SMO	0	0.996	0.039	0.962	0.996	0.957	0.978	0.979
	1	0.961	0.004	0.996	0.961	0.957	0.978	0.978
	Weighted Avg.	0.978	0.022	0.979	0.978	0.957	0.978	0.978
MCC	0	0.999	0.001	0.999	0.999	0.997	0.999	0.999
	1	0.999	0.001	0.999	0.999	0.997	0.999	0.999
	Weighted Avg.	0.999	0.001	0.999	0.999	0.997	0.999	0.999
SGD	0	0.997	0.018	0.982	0.997	0.979	0.989	0.989
	1	0.982	0.003	0.997	0.982	0.979	0.989	0.989
	Weighted Avg.	0.989	0.011	0.989	0.989	0.979	0.989	0.989

Root relative squared error (RRSE): the RRSE squared error takes the total squared error and normalizes it by dividing by the total squared error of the naive model. The RRSE is expressed in percentage. The lower value of RRSE indicates the better performance of the model. The root relative squared error is calculated using the following equation:

$$\text{RRSE} = \sqrt{\sum_{i=0}^{n}\left(\theta_p - \theta_t\right)^2 / \left(\overline{\theta} - \theta_t\right)^2}, \quad (11)$$

where θ_t = true value, θ_p = predicted value (aspected value), and n = total instance, and $\overline{\theta}$ is a mean value of θ.

6. Results and Discussion

This section evaluates the performance of all employed ML techniques. This section is divided into five subsections: the first subsection presents the obtained confusion matrix of all the employed techniques. The second subsection describes the performance under the correctly and incorrectly classified precision and recall rate criteria. The third subsection evaluates the performance-based time and accuracy. The fourth subsection assesses the performance using FPR, ROC, and MCC criteria. Fifth subsection deals with the various MAE, RMSE, RAE, and RRSE error criteria.

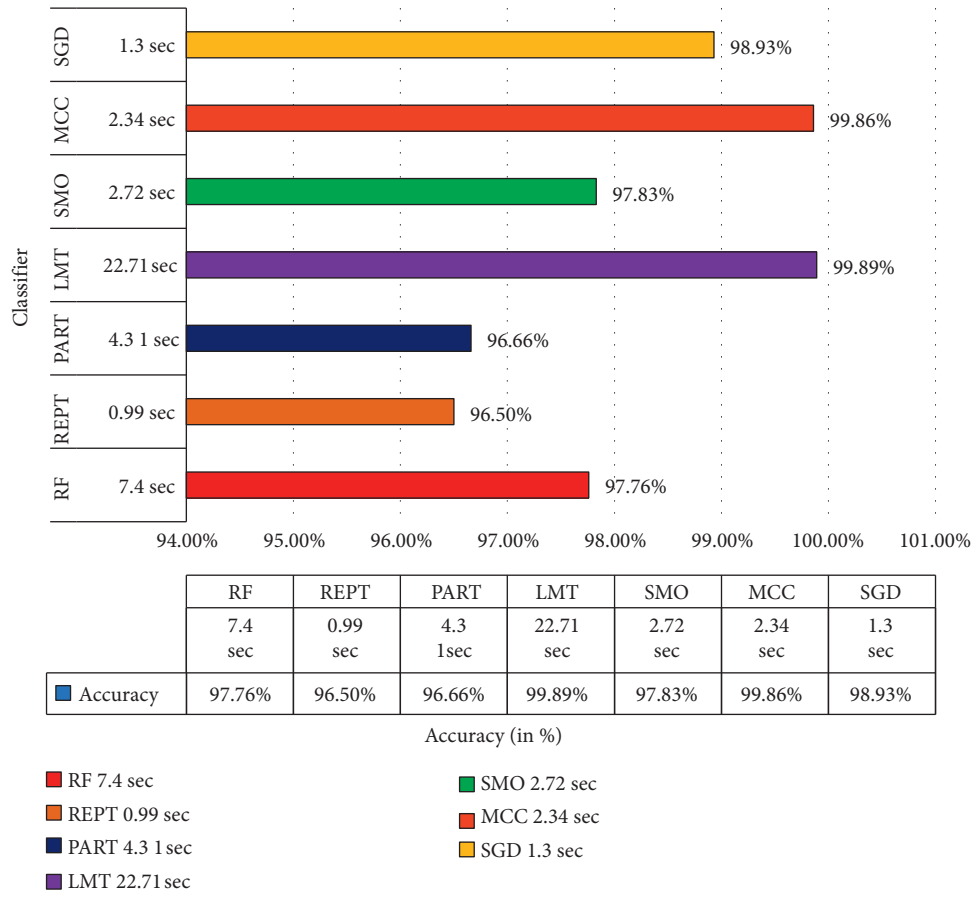


Figure 4: Comparison of RAE and RRSE.

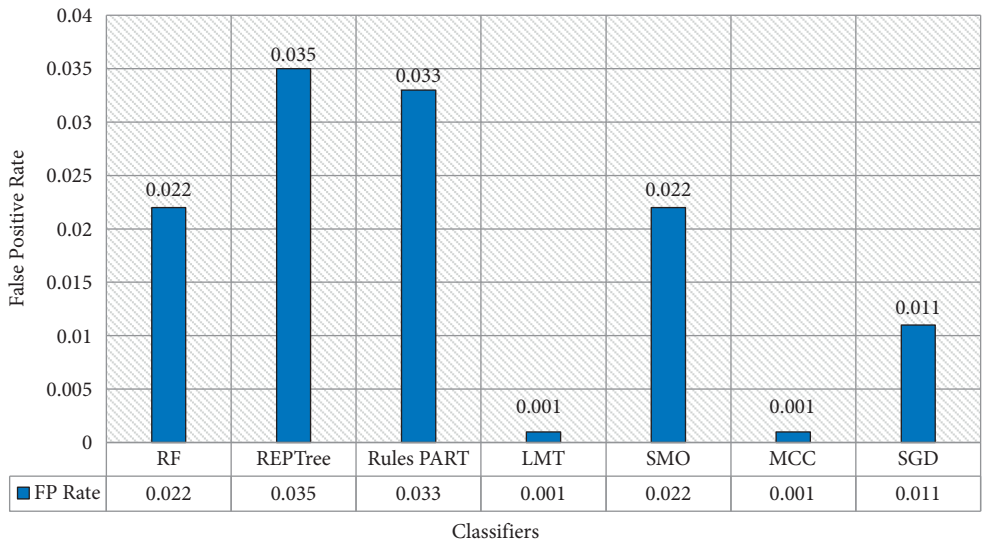
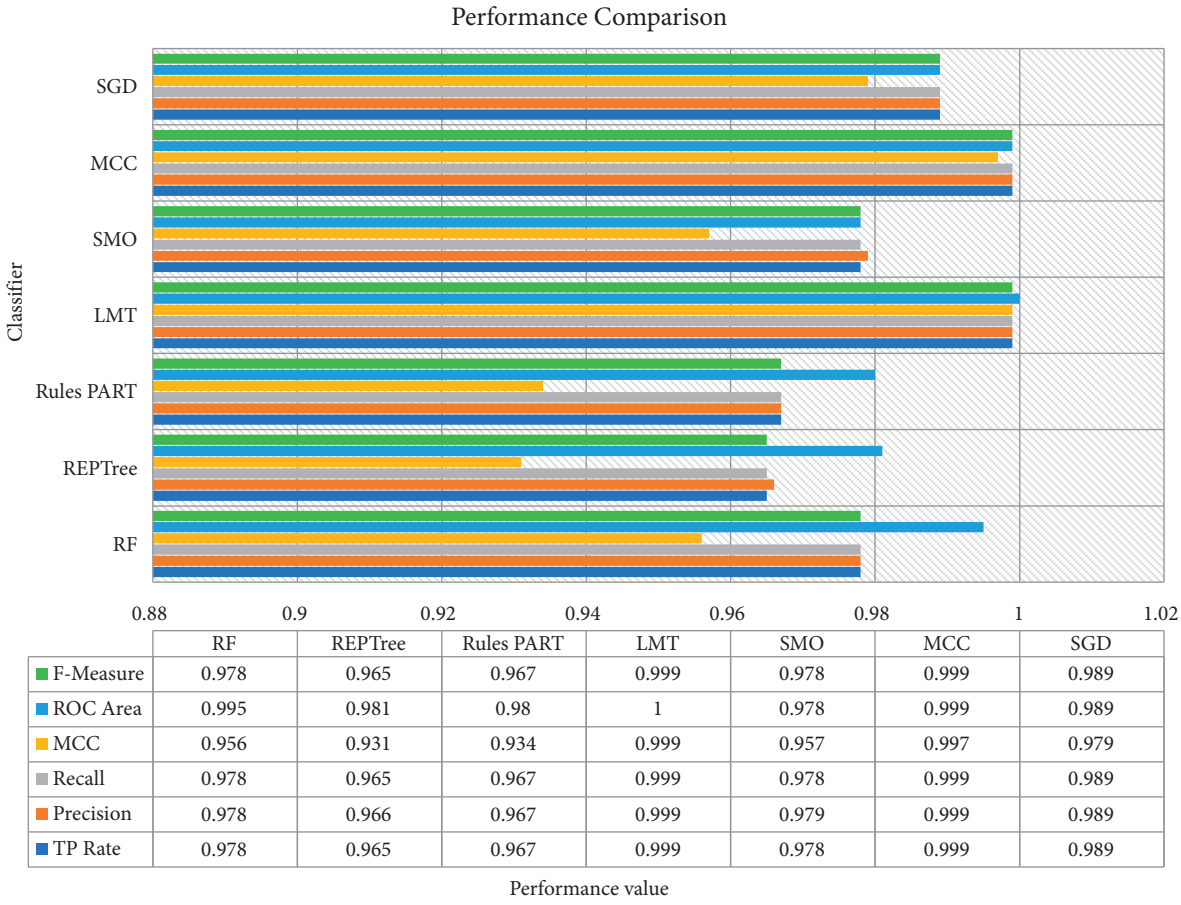


Figure 5: Accuracy of classifiers and time taken to build a model.

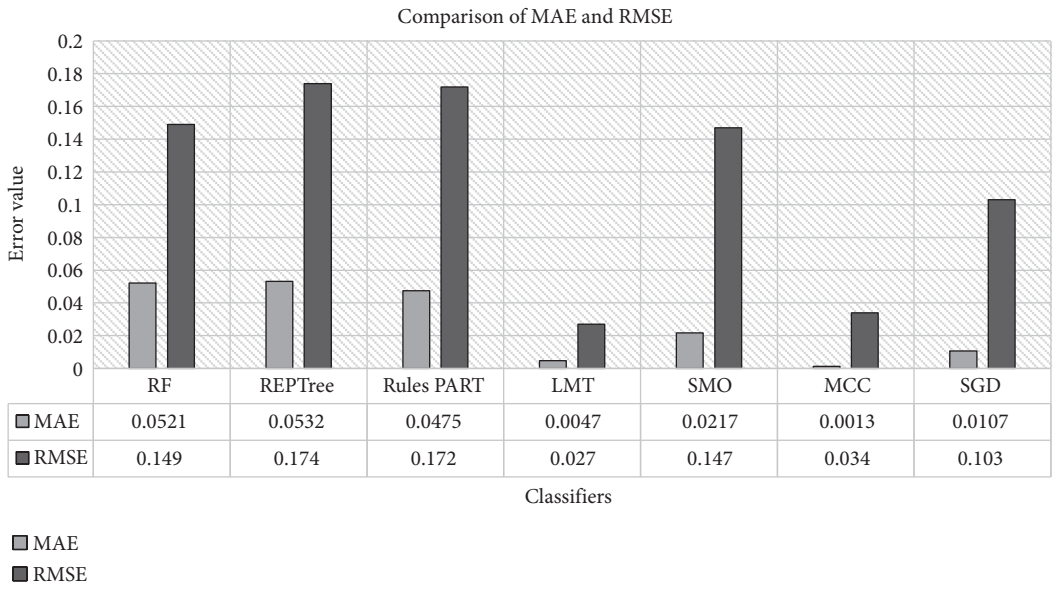
6.1. Confusion Matrix. This subsection presents the derived confusion matrices from all employed ML techniques.

6.2. Evaluation Based on Correctly and Incorrectly Classified Instances, Precision, and Recall Rate. The confusion matrix generated by each model is given in Tables 2–8. Figure 3 depicts the correctly and incorrectly classified instances for each model. The precision and recall values for each classifier are shown in Table 9 and Figure 4. The LMT model correctly classifies 2998 instances, while two are incorrectly classified



	RF	REPTree	Rules PART	LMT	SMO	MCC	SGD
F-Measure	0.978	0.965	0.967	0.999	0.978	0.999	0.989
ROC Area	0.995	0.981	0.98	1	0.978	0.999	0.989
MCC	0.956	0.931	0.934	0.999	0.957	0.997	0.979
Recall	0.978	0.965	0.967	0.999	0.978	0.999	0.989
Precision	0.978	0.966	0.967	0.999	0.979	0.999	0.989
TP Rate	0.978	0.965	0.967	0.999	0.978	0.999	0.989

Figure 6: Comparison of false-positive rate of different classifiers.



	RF	REPTree	Rules PART	LMT	SMO	MCC	SGD
MAE	0.0521	0.0532	0.0475	0.0047	0.0217	0.0013	0.0107
RMSE	0.149	0.174	0.172	0.027	0.147	0.034	0.103

Figure 7: Visualization of performance matrix F-measure, ROC area, MCC, recall, precision, and TP rate.

as malware. The LMT model also demonstrates more significant than 99 percent precision and recall rate, indicating that this approach has the best performance based on these criteria.

6.3. Evaluation of Results Based on Time-to-Build and Accuracy. The REPTree method is the fastest in time-to-build, taking 0.99 seconds, while the LMT method is the slowest, taking 22.71 seconds. As shown in Figure 5, the LMT

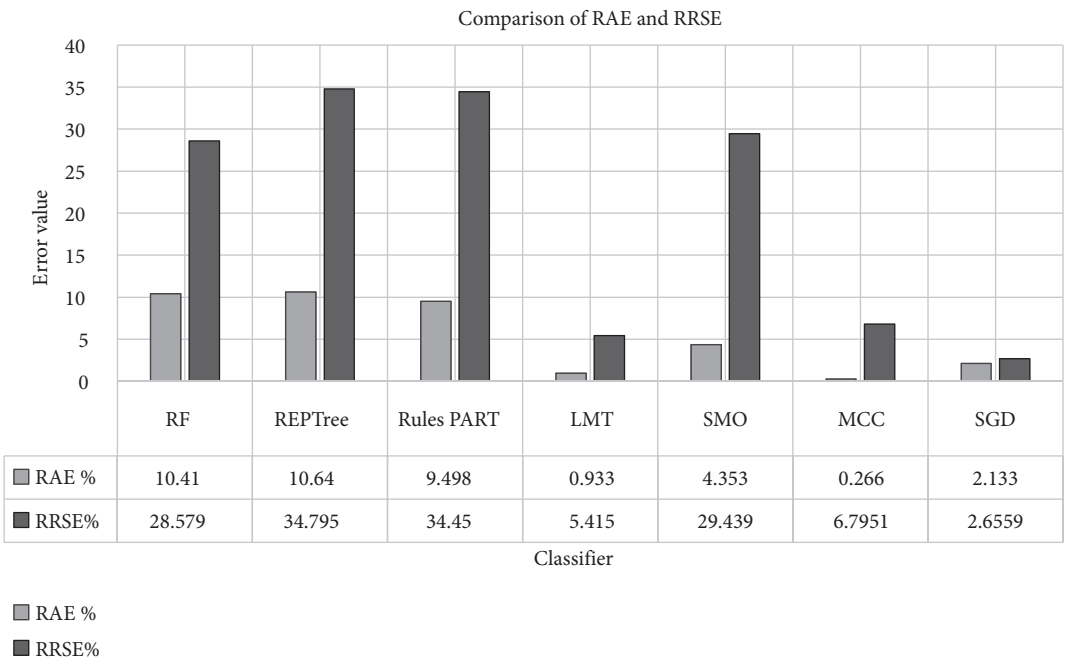


Figure 8: Comparison of MAE and RMSE.

and multiclass classifiers perform best in terms of classification accuracy, with an accuracy of 99.9%. By contrast, REPTree performs poor, with an accuracy of 96.5%, which is the lowest of all applied approaches.

6.4. Evaluation Based on FPR, ROC, and MCC Criteria. The FPR values of the LMT and multiclass are 0.001, which is lower than the all applied approaches. The LMT classifier outperforms all other models in terms of ROC, while the MCC criterion is multiclass classifier and LMT outperform all other models. The FPR values are depicted in Figure 6, while the ROC and MCC values are described in Table 9 and Figure 7.

6.5. Evaluation Based on MAE, RMSE, RAE, and RRSE Error Criteria. This subsection evaluates ML techniques based on error values obtained from all applied classifiers. The errors obtained from all the classifiers are outlined in Figure 4 and 8 respectively. Figure 7 depicts MAE and RMSE, and Figure 8 shows RAE and RRSE. According to the error criterion, the performance of MCC and LMT is the best, as both have the lowest error rate. By contrast, REPTree has the poor performance because its errors are higher than all the classifiers.

7. Conclusion

This paper presented an Android malware detection using a lightweight algorithm (CSF + BFS) for optimal feature selection. Whereas correlation-based feature selection (CFS) evaluates the value of a subset of attributes by considering each feature's predictive ability and the degree of redundancy between them, Best First searches the space of attribute subsets using greedy hill-climbing with a backtracking facility. The number of consecutive non-improving nodes allowed determines the amount of backtracking performed. Thus, this hybrid approach takes the advantage of both CFS and BFS, and the results demonstrate the promising behavior of the proposed CFSBFDroid framework. The proposed algorithm resulted in high classification accuracy, low computational complexity, and quick convergence. The performance of the CFSBFDroid is better than the results reported in the literature. The highest detection accuracy was 99.89%, and the highest obtained F1 score was 99.9%. In the evaluation of precision, recall, and MCC metrics, the proposed approach has achieved a more than 99% score. The proposed model takes 2.34 seconds to build the model, which gives a very low false alarm rate of 0.001. In the future work, we will extend our work to implement some intelligent techniques for Android malware familial classification.

Data Availability

The data are available at https://205.174.165.80/CICDataset/CICMalAnal2017/Dataset/APKs/.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors are very grateful for providing the resources to carry out this work by the Dept. of Computer Science and Applications, Makhanlal Chaturvedi University, Bhopal, M.P, India.

References

[1] G. Play Store, "Number of apps 2021 | Statista," 2021.

[2] W. Ahmed, A. Rasool, A. R. Javed et al., "Security in next generation mobile payment systems: a comprehensive survey," *IEEE Access*, vol. 9, Article ID 115932, 2021.

[3] P. K. R. Maddikunta, Q. V. Pham, B. Prabadevi et al., "Industry 5.0: a survey on enabling technologies and potential applications," *Journal of Industrial Information Integration*, vol. 26, Article ID 100257, 2022.

[4] C. Iwendi, Z. Jalil, A. R. Javed et al., "KeySplitWatermark: zero watermarking algorithm for software protection against cyber-attacks," *IEEE Access*, vol. 8, Article ID 72650, 2020.

[5] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, "Malware detection based on deep learning of behavior graphs," *Mathematical Problems in Engineering*, vol. 2019, pp. 1–10, Article ID 8195395, 2019.

[6] A. Ananya, A. Aswathy, T. R. Amal, P. G. Swathy, P. Vinod, and S. Mohammad, "SysDroid: a dynamic ML-based android malware analyzer using system call traces," *Cluster Computing*, vol. 23, no. 4, pp. 2789–2808, 2020.

[7] S. Lou, S. Cheng, J. Huang, and F. Jiang, "Tfdroid: android malware detection by topics and sensitive data flows using machine learning techniques," in *Proceedings of the 2019 IEEE 2nd International Conference on Information and Computer Technologies, ICICT*, pp. 30–36, Hawaii, HI, USA, March 2019.

[8] R. Taheri, M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti, "Similarity-based Android malware detection using Hamming distance of static binary features," *Future Generation Computer Systems*, vol. 105, pp. 230–247, 2020.

[9] Q. Wu, M. Li, X. Zhu, and B. Liu, "MVIIDroid: a multiple view information integration approach for android malware detection and family identification," *IEEE Multimedia*, vol. 27, no. 4, pp. 48–57, 2020.

[10] J. Feng, L. Shen, Z. Chen, Y. Wang, and H. Li, "A two-layer deep learning method for android malware detection using network traffic," *IEEE Access*, vol. 8, Article ID 125786, 2020.

[11] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[12] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible android malware detection and family classification using network-flows and API-calls," in *Proceedings of the - International Carnahan Conference on Security Technology*, Chennai, India, October 2019.

[13] K. Santosh Jhansi, S. Chakravarty, and R. K. P. Varma, "Feature selection and evaluation of permission-based android malware detection," in *Proceedings of the 4th International Conference on Trends in Electronics and Informatics, ICOEI*, pp. 795–799, Tirunelveli, India, Jun. 2020.

[14] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: deep learning based android malware detection using real devices," *Computers & Security*, vol. 89, Article ID 101663, 2020.

[15] R. Chen, Y. Li, and W. Fang, "Android malware identification based on traffic analysis," in *Lecture Notes in Computer Science*vol. 11632, , pp. 293–303, LNCS, 2019.

[16] X. Jiang, B. Mao, J. Guan, and X. Huang, "Android malware detection using fine-grained features," *Scientific Programming*, vol. 2020, Article ID 5190138, 13 pages, 2020.

[17] R. Thangaveloo, W. Wang Jing, C. Kang Leng, and J. Abdullah, "DATDroid: dynamic analysis technique in android malware detection," *International Journal of Advanced Science, Engineering and Information Technology*, vol. 10, no. 2, pp. 536–541, 2020.

[18] Y. Yang, Z. Wei, Y. Xu, H. He, and W. Wang, "DroidWard : An Effective Dynamic Analysis Method for Vetting Android Applications," *Cluster Computing*, vol. 21, 2016.

[19] A. Amin, A. Eldessouki, M. T. Magdy, N. Abdeen, H. Hindy, and I. Hegazy, "Androshield: automated android applications vulnerability detection, a hybrid static and dynamic analysis approach," *Information*, vol. 10, no. 10, pp. 1–16, 2019.

[20] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: detecting android malware by ORGB analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, 2020.

[21] M. Ahmad, V. Costamagna, B. Crispo, F. Bergadano, and Y. Zhauniarovich, "StaDART: addressing the problem of dynamic code updates in the security analysis of android applications," *Journal of Systems and Software*, vol. 159, 2020.

[22] J. Gajrani, U. Agarwal, V. Laxmi et al., "EspyDroid+: precise reflection analysis of android apps," *Computers & Security*, vol. 90, 2020.

[23] A. I. Ali-Gombe, B. Saltaformaggio, J. R. Ramanujam, D. Xu, and G. G. Richard, "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming," *Computers & Security*, vol. 73, pp. 235–248, 2018.

[24] H. Zhu, Y. Li, R. Li, J. Li, Z. You, and H. Song, "SEDMDroid: an enhanced stacking ensemble framework for android malware detection," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 984–994, 2021.

[25] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 6, pp. 712–736, 2018.

[26] F. Martinelli, F. Mercaldo, and A. Saracino, "Bridemaid: an hybrid tool for accurate detection of android malware," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 899–901, Abu Dhabi, UAE, April 2017.

[27] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting android malware," *Computer Networks*, vol. 167, Article ID 107026, 2020.

[28] K. Sugunan, T. Gireesh Kumar, and K. A. Dhanya, "Static and dynamic analysis for android malware detection," *Advances in Intelligent Systems and Computing*, vol. 645, pp. 147–155, 2018.

[29] A. Martín, V. Rodríguez-Fernández, and D. Camacho, "CANDYMAN: classifying Android malware families by modelling dynamic traces with Markov chains," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 121–133, 2018.

[30] Y. Yang, Z. Wei, Y. Xu, H. He, and W. Wang, "Droidward: an effective dynamic analysis method for vetting android applications," *Cluster Computing*, vol. 21, no. 1, pp. 265–275, 2018.

[31] R. Surendran, T. Thomas, and S. Emmanuel, "A TAN based hybrid model for android malware detection," *Journal of Information Security and Applications*, vol. 54, Article ID 102483, 2020.

[32] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, 2019.

[33] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, Article ID 10614, 2020.

[34] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: mining patterns of malicious and benign android apps via permission-related APIs," *IEEE Transactions on Reliability*, vol. 67, no. 1, pp. 355–369, 2017.

[35] S. Garg and N. Baliyan, "A novel parallel classifier scheme for vulnerability detection in android," *Computers & Electrical Engineering*, vol. 77, pp. 12–26, 2019.

[36] A. Maryam, A. Usman, M. Aleem, J. C.-W. Lin, M. A. Islam, and M. A. Iqbal, "cHybriDroid: a machine learning based hybrid technique for securing the edge computing," *Security and Communication Networks*, vol. 2020, Article ID 8861639, 14 pages, 2020.

[37] N. Viet Duc and P. Thanh Giang, "NADM: neural network for android detection malware," in *Proceedings of the Ninth International Symposium on Information and Communication Technology (SoICT 2018)*, pp. 449–455, Association for Computing Machinery, Danang City, Vietnam, December 2018.

[38] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and API calls," *Future Generation Computer Systems*, vol. 107, pp. 509–521, 2020.

[39] V. Kumar, "Evaluation of computationally intelligent techniques for breast cancer diagnosis," *Neural Computing & Applications*, vol. 33, no. 8, pp. 3195–3208, 2021.

[40] A. D. Amirruddin, F. M. Muharam, M. H. Ismail, N. P. Tan, and M. F. Ismail, "Synthetic Minority Over-sampling TEchnique (SMOTE) and Logistic Model Tree (LMT)-Adaptive Boosting algorithms for classifying imbalanced datasets of nutrient and chlorophyll sufficiency levels of oil palm (Elaeis guineensis) using spectroradiometers and unmanned aerial vehicles," *Computers and Electronics in Agriculture*, vol. 193, Article ID 106646, 2022.

[41] S. I. Imtiaz, S. U. Rehman, A. R. Javed, Z. Jalil, X. Liu, and W. S. Alnumay, "DeepAMD: detection and identification of android malware using high-efficient deep artificial neural network," *Future Generation Computer Systems*, vol. 115, pp. 844–856, 2021.

[42] E. Odat, B. Alazzam, and Q. M. Yaseen, "Detecting malware families and subfamilies using machine learning algorithms: an empirical study," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 2, pp. 761–765, 2022.

[43] CIC Dataset, "Index of/CICDataset/CICMalAnal2017/Dataset/APKs," 2022.

[44] M. J. Niranjan, *Apktool - A Tool for Reverse Engineering 3rd Party, Closed, Binary Android Apps*, 2020, https://ibotpeaches.github.io/Apktool/.

[45] Google, *AAPT2 | Android Developers*, 2019, https://developer.android.com/studio/command-line/aapt2.