sustainability

MDPI

Article

Android Malware Classification Using Optimized Ensemble Learning Based on Genetic Algorithms

Altyeb Taha * and Omar Barukab

Department of Information Technology, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Jeddah 21911, Saudi Arabia
* Correspondence: aaataha@kau.edu.sa

Abstract: The continuous increase in Android malware applications (apps) represents a significant danger to the privacy and security of users' information. Therefore, effective and efficient Android malware app-classification techniques are needed. This paper presents a method for Android malware classification using optimized ensemble learning based on genetic algorithms. The suggested method is divided into two steps. First, a base learner is used to handle various machine learning algorithms, including support vector machine (SVM), logistic regression (LR), gradient boosting (GB), decision tree (DT), and AdaBoost (ADA) classifiers. Second, a meta learner RF-GA, utilizing genetic algorithm (GA) to optimize the parameters of a random forest (RF) algorithm, is employed to classify the prediction probabilities from the base learner. The genetic algorithm is used to optimize the parameter settings in the RF algorithm in order to obtain the highest Android malware classification accuracy. The effectiveness of the proposed method was examined on a dataset consisting of 5560 Android malware apps and 9476 goodware apps. The experimental results demonstrate that the suggested ensemble-learning strategy for classifying Android malware apps, which is based on an optimized random forest using genetic algorithms, outperformed the other methods and achieved the highest accuracy (94.15%), precision (94.15%), and area under the curve (AUC) (98.10%).

Keywords: Android malware classification; genetic algorithms; random

Citation: Taha, A.; Barukab, O. Android Malware Classification Using Optimized Ensemble Learning Based on Genetic Algorithms. *Sustainability* **2022**, *14*, 14406. https://doi.org/10.3390/su142114406

Academic Editor: Andreas Kanavos

Received: 21 September 2022
Accepted: 1 November 2022
Published: 3 November 2022

1. Introduction

Android is an open-source smartphone operating system (OS) that dominated the global smartphone OS shipment market in 2021, accounting for 83.8% of the overall market [1]. Android applications have been increasingly utilized in many aspects of daily life. Because Android-based smartphones are used by thousands of people every year, cyber-criminals are constantly developing dangerous applications in order to steal sensitive information and perform malicious attacks. According to the Kaspersky mobile threats report in 2021 [2], cyber-criminals have increased the effectiveness of their mobile attacks by developing more complex mobile malware that utilizes new methods to steal users' sensitive information. According to the study, 3.5 million malware apps were discovered, resulting in 46.2 million cyberattacks worldwide in 2021. Because the amount and complexity of malware are rapidly increasing, it is critical to develop effective malware classification approaches to handle this issue [3,4]. However, malicious software developers typically make minor changes to the malware app's original source code, creating new, dangerous software variants in order to evade malware detection systems. As a result, identifying new malicious software variants becomes more difficult [5].

To identify and categorize Android malware, security researchers have employed different approaches that can be classified as static, dynamic, or mixed, and they are all done with the help of machine learning [6]. Static analysis parses the source code of an Android Application Package (APK) file without running it. APK files include numerous features, including API calls, network addresses, permissions, hardware structures, and

signatures [7]. Each of these characteristics can be utilized to determine the maliciousness of an Android application. Static analysis is relatively inexpensive; however, it is unable to recognize malware attacks that use zero-day vulnerabilities [8]. Dynamic analysis, on the other hand, investigates the app's features and behavior during operation. It isolates the app from the outside world by placing it in a sandbox and observing its behavior; when an application's behavior is found to not follow the established pattern, it is classified as malware. Various behavior-based characteristics, such traffic in network traffic [9], API calls [10], and log files of the system [11], can be obtained during dynamic analysis. However, because the analysis is carried out in an isolated setting, the malware's behavior may vary during its execution. As a result, dynamic analysis may fall short of catching the true activities of a malware attack.

Although malware detection using a single machine-learning classifier has been thoroughly investigated, the performance of each model differs because of variations in training datasets and strategies of feature selection. Moreover, each classifier has inherent limitations and uncertainty. Using a combination of classifiers has an advantage over using a single classifier in reducing the variance of expected error and enhancing classification accuracy. Ensemble learning is a machine learning strategy that employs several methods instead of relying on a single method to build a single accurate classifier; it has been proven to be a successful approach in a range of fields. Theoretically and practically, it has been demonstrated that ensemble learning approaches outperform weak single methods, especially when solving complex and high-dimensional prediction problems [12]. The most prevalent ensemble learning techniques are bagging [13], boosting [14], and stacking [15]. Stacking is a merging strategy that fuses numerous machine learning methods (base models) organized in one stage and then applies a different machine learning method (meta model) to obtain a more accurate classification model [16].

An ensemble learning method can be developed to address the shortcomings of single-model-based methods. It combines numerous separate classifiers and typically outperforms a single model in terms of generalization performance. In this paper, an approach for Android malware classification using optimized ensemble learning based on genetic algorithms is proposed. The proposed approach consists of two stages. First, a base learner is constructed to handle different machine learning algorithms, including support vector machine (SVM), logistic regression (LR), K-nearest neighbor (KNN), decision tree (DT), gradient boosting (GB), and AdaBoost (ADA) classifiers. Second, a meta learner RF-GA, utilizing genetic algorithms (GAs) to optimize the parameters of the random forest (RF) algorithm, is employed to classify the prediction probabilities from the base learner. This optimization mechanism involves setting the parameters in the RF algorithm training procedure, which improves the accuracy of the differentiation between goodware and malware apps.

The contributions of this research are as follows:

- An improved ensemble learning approach based on genetic algorithms is proposed to classify Android malware apps.
- In order to increase Android malware classification accuracy, genetic algorithms are used to optimize the meta learner's parameters.
- In comparison to other techniques, the suggested method achieved the highest classification accuracy.

The rest of the paper is organized as follows: Section 2 discusses works relevant to Android malware classification; Section 3 introduces the proposed approach for Android malware classification using optimized ensemble learning based on genetic algorithms; Section 4 presents the experimental results and discussion; finally, Section 5 concludes this paper and presents possible future work.

2. Literature Review

Several approaches have been suggested for Android malware classification based on machine learning techniques. Machine learning (ML) is a technique in which a system learns a pattern, builds a model, and provides classifications based on the input data.

SEDMDroid, a stacking integration framework proposed by Zhu et al. [17], is a tool for detecting Android malware. By obtaining all principal components and utilizing the complete dataset to train each base learner multilayer perceptron (MLP), the analysis of the principal component was performed on each feature subset. Then, as a fusion classifier, the support vector machine (SVM) algorithm was employed to learn hidden supplemental data from the results of ensemble models and generate the ultimate classification. Using multi-level static features, the suggested method achieved an accuracy of 89.07%. Idress et al. [18] suggested PIndroid, a novel permission- and intent-based approach for detecting malicious apps. PIndroid was one of the pioneer methods for identifying malware accurately by combining permissions and intent, supplemented by an ensemble technique. The authors investigated the link between permissions and intent using statistical significance tests, and found a statistically significant, robust association between intent and permissions that could be used to classify malware apps.

Rana et al. [19] suggested and analyzed multiple machine learning techniques for classifying Android malware, coupled with a substring-based classifier feature selection (SBFS) strategy using an ensemble-based learning methodology. They used the Drebin dataset and an ensemble learning approach to achieve better results. Li et al. [20] proposed an Android malware classification approach based on permissions. This approach utilized 22 Android permissions to differentiate between goodware and malware apps based on ML algorithms. The dataset of permissions was generated using a database of benign and malware apps from Google Play. The support vector machine (SVM) performed better than the other studied ML algorithms (decision tree (DT) and naïve Bayes (NB)) and achieved the highest accuracy of 90%. Lou et al. [21] suggested a machine learning method to differentiate between Android goodware and malware apps. The proposed method employed the SVM algorithm for the detection of Android malware. The Drebin and Google Play datasets were utilized to train the classifier, and the highest accuracy achieved by the proposed approach was 93.7%. Since this study used SVM only, it is preferable to examine the performance of more ML algorithms.

Firdaus et al. [22] suggested a method for detecting Android malware apps based on feature analysis, with Android permissions and directory paths among the features. With this method, based on the AndroZoo and Drebin datasets, they utilized three machine learning methods: multilayer perceptron (MLP), radial basis function network (RBFN), and voted perceptron (VP). Their proposed MLP-based technique had a classification accuracy of 90% and an 87% true positive rate. Arp et al. [23] developed "Drebin", software for identifying malicious Android applications. To distinguish between good and malicious programs, they employed network addresses, Android permissions, and API calls as characteristics. Their findings were based on a dataset that included 123,453 goodware and 5560 malware apps from a variety of app stores. Drebin attained 94% detection accuracy and a 1% false positive rate.

We proposed an evolving hybrid neuro-fuzzy classifier (EHNFC) for Android malware classification, utilizing characteristics based on permissions in a prior study [24]. We improved the evolving clustering algorithm to include a dynamic mechanism for adjusting clustered permission-based feature radii and centers. The results show that the suggested technique diagnoses Android malware with an accuracy of 90%. In a former study [25], we presented a strategy for Android malware classification utilizing adaptive neuro-fuzzy inference systems (ANFIS), an information gain algorithm used to select the necessary Android permissions. The suggested approach achieved 75% accuracy. We introduced an adaptive neuro-fuzzy inference method with fuzzy c-means clustering (FCM-ANFIS) for the classification of Android malware in our previous research [26]. According to the results of the experiments, the proposed technique attained classification accuracy of 91%.

In [27], Garg and Niyati proposed an approach for Android malware classifiers based on parallel classifiers. Their proposed methodology merged characteristics from several parallel classifiers using expectation maximization and achieved 98.27% accuracy.

3. Suggested Approach

The suggested approach is an ensemble method for classifying malware/goodware that uses a random forest classifier (meta learner) to make the final decision using the predictions of other ensemble members (base learner) as features. The base learner consists of six classifiers: gradient boosting (GB), AdaBoost (ADA), K-nearest neighbor (KNN), logistic regression (LR), support vector machine (SVM), and decision tree (DT). To achieve the maximum classification accuracy for Android malware apps, genetic algorithms were utilized to enhance the base learner's parameters. The suggested approach is depicted in Figure 1.

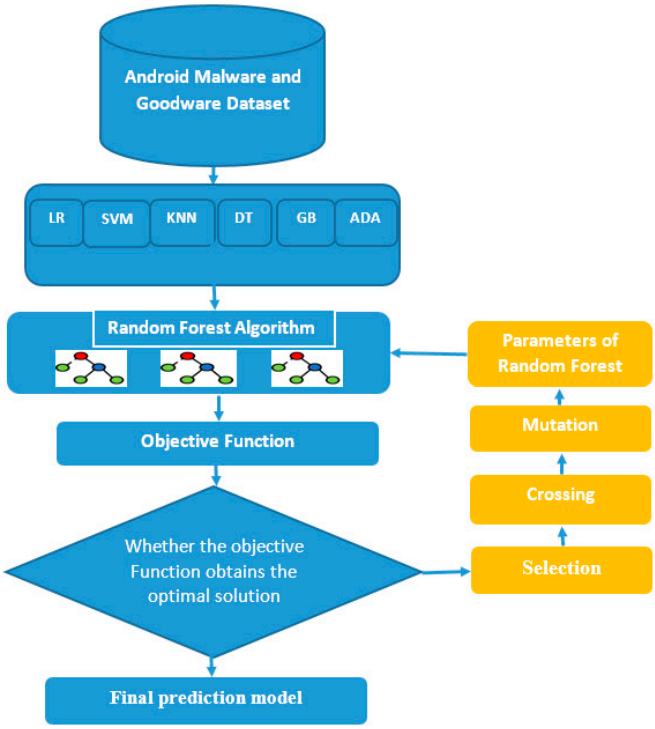

Figure 1. Proposed approach for Android malware classification.

3.1. Dataset and Feature Selection

A dataset of 5560 Android malware apps and 9476 Android goodware apps was used in this research [23]. The malware apps in the dataset came from 49 distinct malware families, such as Goldmaster, DroidKungFu, GingerMaster, and FakeInstaller. New Android app samples were included in the dataset. In preparation for evaluation, preprocessing was performed on the dataset, and it was divided into training and testing datasets. Android permissions were included in the dataset as a differentiating factor between good and malicious apps.

Feature selection methods can help to increase the accuracy of classification by employing key characteristics that define the distinctive properties of the dataset. These methods shrink datasets by omitting the permissions that are not essential for categorization. Machine learning algorithms might suffer from redundancy or dissimilar properties, which can cause a range of problems, including a decline in their efficacy. It is important to employ a suitable feature selection strategy in order to attain precise detection of Android malware apps.

To boost classification accuracy, the crucial Android permissions were specified in this research using the information gain ratio (IGR) method [28]. The IGR gives the maximum

score to the most potential permissions based on the class of malware or goodware the Android app belongs to. The IGR equations are described as follows:

$$gain\_r(X, C) = \frac{gain\_r(P, M)}{split\_info(M)} \quad (1)$$

$$split\_info(C) = \sum_i \left(\frac{|M_i|}{|M|} \right) log \frac{|M_i|}{|M|} \quad (2)$$

where *gain_r*(*P*,*M*) indicates the gain ratio of permission *P* occurrence in class and *M*, *Mi* and $|Mi|$ denote the occurrence of permission *P* in class *M*, the *i*th subclass of *M*, and the total number of permissions in *Mi*, respectively.

The IGR algorithm depends on determining the relations between an Android app's permissions and then calculating and rating each permission separately. Because of its great efficiency and quick calculation, the information-gain ratio is used to choose the key Android app permissions in the proposed method. It evaluates the appropriate permissions based on class (malware or goodware). Figure 2 shows some of the features included in the proposed model, as well as their information gain rankings. The horizontal axis displays the information gain ranks for the chosen permissions.

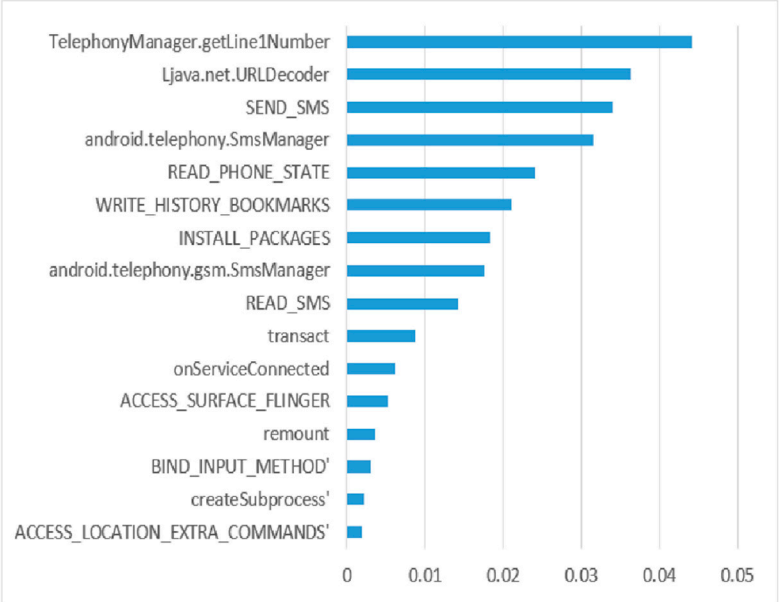

Figure 2. Selected features for Android malware classification.

3.2. Random Forest Algorithm

A random forest classifier uses an ensemble approach to make predictions by generating and using many classification trees. Using a statistical approach known as bootstrapping, the classification trees are trained independently on a portion of the training data. Bootstrapping is a strategy for obtaining a large number of datasets by sampling from the original dataset. The sampling can be done with replacement, which means that a sampled subset can contain duplicated data points, allowing for a high number [29].

3.3. Genetic Algorithms

The genetic algorithm (GA) is based on the evolutionary hypothesis that individuals with greater capacity for environmental adaptation are more likely to survive and consequently pass on their abilities to the next generation. As the children inherit their parents' features, they may generate individuals who are better or worse replicas of their parents [30]. Better individuals will have a higher chance of surviving and, as a result, having better offspring, whereas the worst ones will eventually disappear. The individual with

the top chromosomes (properties) will be selected as the global optimum after numerous generations [31]. Population size, representing the total number of solutions, is one of the most essential elements of GA and has a major impact on the task results, as the number of generations corresponds to the maximum number of iterations of the algorithm [32]. GA employs the probabilistic optimization method to obtain the optimization search space automatically and adapt the search path as needed. There are three steps in the basic process of selecting good variations, mutations, and crossings:

1. Generate a chromosome population.
2. Use the fitness function to assess the fitness value of individuals in the population.
3. Create new populations using genetic operators (selection, crossover, and mutation).
4. Repeat steps 2 and 3 indefinitely until the termination requirements are met.

In the process of using GA to optimize the RF algorithm parameters in our study, it was essential to define the GA parameters, which included population size (μ), iteration number (I), mutation_rate, crossover_rate, and fitness function. In this work, we chose μ = 100, I = 100, crossover_rate = 0.1, and mutation_rate = 0.9, as these values obtained the best results. For the fitness function, the fitness of each population chromosome was assessed based on classification accuracy.

3.4. Optimization of Random Forest Using GA

In the suggested ensemble learning technique, the genetic algorithm was employed to optimize the parameters of the base leaner, which is the RF algorithm, by identifying RF parameters that enhanced the accuracy of Android malware classification. The inputs to the random forest classifier were the prediction probabilities resulting from the base learner and the hyperparameters obtained from the GA, as shown in Figure 1. The GA-RF implementation steps are as follows:

Step 1: Initialize the population. All chromosomes in the initial population are binary-coded, and there is a certain number of chromosomes chosen at random. A gene that is encoded 0 means that a certain characteristic is not desired.

Step 2: Assess each entity's fitness. Entities with a greater fitness value are nearer to the algorithm's ideal outcome. This hybrid model's fitness function is based on the average accuracy using five-fold cross-validation.

Step 3: Apply GA operators. A parent population for subsequent procedures is created by first screening out numerous pairs of chromosomes based on the fitness of various entities using the selection operator. Entities in the parent population then carry out crossover and mutation operations using the respective operator defined in GA to create candidate entities with new attributes.

Step 4: Output hybrid model. Repeat step 3 until the maximum number of iterations is attained. The model outputs the ideal RF parameters when the stop criterion is fulfilled.

GA generates a candidate hyperparameter configuration in each of its allotted iterations, then these parameters are used along with the prediction probabilities from the base learner to obtain the classification accuracy of the trained model. The configuration that produced the best classification accuracy was utilized to train the final model before it was tested on the test dataset, yielding the value of random forest classification accuracy when utilizing GA. Table 1 shows the RF parameters optimized by GA.

Table 1. Optimized parameters used in meta leaner with GA.

Hyperparameter	Explanation	Range	Best Value
Max_depth	Greatest depth of decision tree	10–100	90
Min_samples_leaf	Lowest number of samples required to be a leaf node	1–10	1
Min_samples_split	Lowest number of samples required to split an internal node	1–10	5
n_estimators	Number of trees in forest	100–1000	800

3.5. Performance Measure

A confusion matrix is commonly used to assess the effectiveness of machine learning techniques. Table 2 provides an example of a confusion matrix, which contains the following information:

Table 2. Example of confusion matrix.

		Predicted Class	
		Negative	**Positive**
Real Class	Negative	True negative (TN)	False positive (FP)
	Positive	False negative (FN)	True positive (TP)

TP: The number of Android apps that are actually malware and are predicted as malware.
TN: The number of Android apps that are actually goodware and are predicted as goodware.
FP: The number of Android apps that are actually goodware but are predicted as malware.
FN: The number of Android apps that are actually malware but are predicted as goodware.

Accuracy is defined as the percentage of correct classifications divided by the total number of classification options. Accuracy can be mathematically described as:

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN)$$

Precision relates to how often the machine learning algorithm is correct. Precision can be expressed mathematically as:

$$\text{Precision} = TP/(TP + FP)$$

Recall is a metric for true positive rates. The mathematical expression for recall is:

$$TP = TP/(TP + FN)$$

The F-measure is defined as the harmonic average of recall and precision:

$$\text{F-measure} = (2 \times \text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall})$$

A receiver operating characteristic (ROC) curve is a graph illustrating the performance of the classifier at all classification thresholds. This curve plots two parameters: false positive rate and true positive rate.

True positive rate (TPR) is another term for recall and is therefore defined as follows:

$$TPR = TP/(TP + FN)$$

False positive rate (FPR) is defined as follows:

$$FPR = FP/(FP + TN)$$

The area under the ROC curve (AUC) is a visual illustration of the FPR and TPR using various folds. AUC has higher accuracy in terms of performance evaluation because it is not predicated on a threshold. A classifier with an AUC value close to 1 generally performs better. AUC, precision, and recall measurements were utilized to assess the performance of the suggested strategy.

Precision and recall are important parameters for assessing classifier performance. Precision is a metric that indicates how many Android applications identified as malware were indeed malware, whereas recall indicates how many of the total relevant results were correctly categorized. A high score for recall implies a low proportion of false negatives (FN), whereas accuracy with a high score suggests a low number of false positives (FP). High percentages of precision and recall indicate that the model correctly provides classification results [33]. As a result, the precision–recall score curve provides a reliable estimate of the classification model's accuracy [34].

4. Results and Discussion

In this section, we evaluate the performance of the proposed approach by comparing it to six basic classifiers using information from the experiments. The basic classifiers were logistic regression (RL), support vector machine (SVM), K-nearest neighbor (KNN), decision tree (DT), gradient boosting (GB), and AdaBoost (ADA).

The suggested technique has the highest accuracy score of 94.15 %, proving its ability to identify malware from good software (Table 3). Furthermore, the suggested method earned a recall score of 94.15%, proving its capacity to correctly identify 94.15% of Android applications while reducing false positives. The RF algorithm attained the second highest accuracy of 93.77%, while AdaBoost achieved the lowest accuracy of 90.96%. The proposed approach attained the highest precision, recall, and F1 scores of 94.153, 94.151, and 94.152%, respectively.

Table 3. Comparison of performance of proposed approach and other machine learning algorithms.

Approach	Accuracy	Precision	Recall	F1
SVM-Testing	0.93620	0.9364	0.9362	0.93628
SVM-Training	0.9417	0.94166	0.9417	0.94168
Decision tree-Testing	0.93753	0.93762	0.93753	0.93757
Decision tree-Training	0.94341	0.9434	0.94341	0.94341
Random forest-Testing	0.93775	0.93793	0.93775	0.93782
Random forest-Training	0.94341	0.94342	0.94341	0.94342
Logistic regression-Testing	0.92867	0.92914	0.92867	0.92883
Logistic regression-Training	0.93031	0.93058	0.93031	0.93041
K-nearest neighbor-Testing	0.93199	0.93273	0.93199	0.93221
K-nearest neighbor-Training	0.93952	0.93971	0.93952	0.93959
Gradient boosting-Testing	0.93354	0.93382	0.93354	0.93365
Gradient boosting-Training	0.93667	0.93675	0.93667	0.93365
AdaBoost-Testing	0.90961	0.90923	0.90961	0.9092
AdaBoost-Training	0.90995	0.90957	0.90995	0.90685
Proposed approach-Testing	0.94152	0.94153	0.94151	0.94152
Proposed approach-Training	0.94474	0.94464	0.94474	0.94466

Figure 3 shows the AUC, which was used to analyze the suggested technique. The AUC is a valuable and relevant indicator of overall performance. A high AUC value suggests a stronger ability to categorize. As shown in Figure 3, the suggested technique has an AUC of 98.1%, demonstrating that it can efficiently differentiate between Android malware and goodware apps, and proving that our proposed ensemble classifier performed better than the classifiers we used as base models.

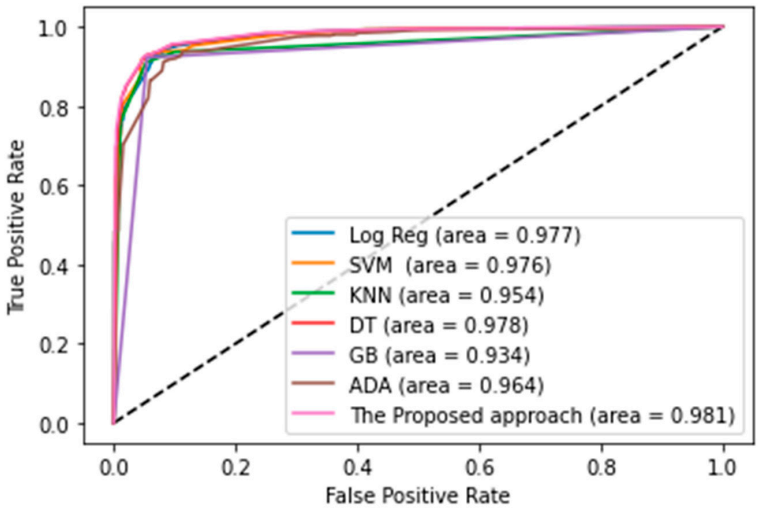

Figure 3. AUC for proposed approach and some machine learning techniques.

In general, the precision–recall curve is used to evaluate classifiers depending on recall and precision, and is visualized on a graph in which the ratio of precision is given on the Y-axis and the recall on the X-axis. The precision–recall curve provides a comprehensive view of classifier evaluation. The precision–recall curve of the proposed classifier was compared to those of existing machine learning techniques, as shown in Figure 4.

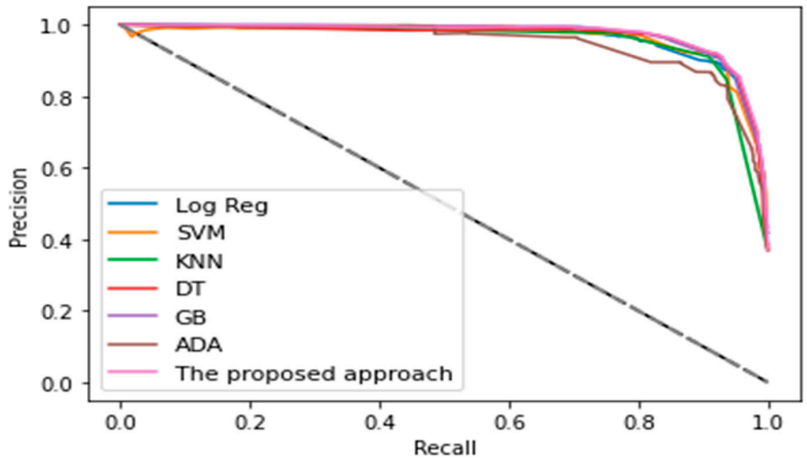

Figure 4. Precision–recall curves of proposed approach and some other machine learning techniques.

The performance of the proposed approach was also compared to that of existing techniques. Table 4 compares the performance of the suggested approach with other approaches.

Table 4. Comparison between suggested approach and prior research.

Approach	Accuracy
Huang et al. [35]	93%
Singh et al. [36]	92.59%
Altaher [24]	90%
Abdulla and Altaher [25]	75%
Altaher and Barukab [26]	91%
Proposed approach	94%

5. Conclusions

Because Android malware applications damage sensitive data, resulting in large financial losses, it is critical to develop efficient categorization methods for them. Using optimized ensemble learning based on genetic algorithms, this study provides a solution for Android malware classification. The suggested method consists of two steps. First, a base learner is constructed to handle various machine learning algorithms, including support vector machine (SVM), logistic regression (LR), K-nearest neighbor (KNN), decision tree (DT), gradient boosting (GB), and AdaBoost (ADA) classifiers. Second, to classify the prediction probabilities from the base learner, a meta learner RF-GA is used, which employs the genetic algorithm (GA) to optimize the parameters of the random forest (RF) algorithm. GAs were utilized to adjust the RF algorithm's parameter settings in order to achieve the highest Android malware classification accuracy. The experimental findings show that the proposed ensemble learning technique for Android malware classification, based on optimized random forest using genetic algorithms, outperformed the other approaches and reached the highest accuracy of 94.15%.

Author Contributions: Conceptualization, A.T.; methodology, A.T.; implementation, A.T.; writing original draft preparation, A.T.; writing—review and editing, O.B. All authors have read and agreed to the published version of the manuscript.

Funding: The Deanship of Scientific Research (DSR) at King Abdulaziz University (KAU), Jeddah, Saudi Arabia has funded this Project under grant no (G: 270-830-1443).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The Deanship of Scientific Research (DSR) at King Abdulaziz University (KAU), Jeddah, Saudi Arabia has funded this Project under grant no (G: 270-830-1443).

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

References

1. Statista. Available online: https://www.statista.com/statistics/1236760/worldwide-smartphone-operating-system-shipment-market-share/#statisticContainer (accessed on 1 June 2022).
2. Kaspersky. Available online: https://usa.kaspersky.com/about/press-releases/2022_2021-mobile-threats-report-cybercriminals-pursue-banking-and-gaming-accounts (accessed on 1 June 2022).
3. Shang, Y. Consensus of Hybrid Multi-Agent Systems With Malicious Nodes. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 685–689. [CrossRef]
4. Fragkos, G.; Minwalla, C.; Plusquellic, J.; Tsiropoulou, E.E. Artificially Intelligent Electronic Money. *IEEE Consum. Electron. Mag.* **2021**, *10*, 81–89. [CrossRef]
5. Selvaganapathy, S.G.; Sadasivam, S.; Ravi, V. A review on Android malware: Attacks, countermeasures and challenges ahead. *J. Cyber Secur. Mobil.* **2021**, *10*, 177–230. [CrossRef]
6. Wu, Q.; Zhu, X.; Liu, B. A Survey of Android Malware Static Detection Technology Based on Machine Learning. *Mob. Inf. Syst.* **2021**, *2021*, 8896013. [CrossRef]
7. Mantoo, B.A.; Khurana, S.S. Static, dynamic and intrinsic features based Android malware detection using machine learning. In *Proceedings of ICRIC*; Springer: Cham, Switzerland, 2020.
8. Hemalatha, J.; Roseline, S.A.; Geetha, S.; Kadry, S.; Damaševičius, R. An efficient DenseNet-based deep learning model for malware detection. *Entropy* **2021**, *23*, 344. [CrossRef]
9. Mohaisen, A.; Alrawi, O.; Mohaisen, M. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Comput. Secur.* **2015**, *52*, 251–266. [CrossRef]
10. Amer, E.; Zelinka, I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* **2020**, *92*, 101760. [CrossRef]
11. Sihwail, R.; Omar, K.; Ariffin, K.Z.; Afghani, S.A. Malware detection approach based on artifacts in memory image and dynamic analysis. *Appl. Sci.* **2019**, *9*, 3680. [CrossRef]
12. Dong, X.; Yu, Z.; Cao, W.; Shi, Y.; Ma, Q. A survey on ensemble learning. *Front. Comput. Sci.* **2019**, *14*, 241–258. [CrossRef]
13. Breiman, L. Bagging predictors. *Mach. Learn.* **1996**, *24*, 123–140. [CrossRef]
14. Schapire, R.E.; Singer, Y. Improved Boosting Algorithms Using Confidence-rated Predictions. *Mach. Learn.* **1999**, *37*, 297–336. [CrossRef]

15. Wolpert, D.H. Stacked generalization. *Neural Netw.* **1992**, *5*, 241–259. [CrossRef]
16. Sagi, O.; Rokach, L. Ensemble learning: A survey. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2018**, *8*, e1249. [CrossRef]
17. Zhu, H.; Li, Y.; Li, R.; Li, J.; You, Z.; Song, H. SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 984–994. [CrossRef]
18. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [CrossRef]
19. 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]
20. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant permission identification for machine-learning-based android malware detection. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3216–3225. [CrossRef]
21. Lou, S.; Cheng, S.; Huang, J.; Jiang, F. 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), Kahului, HI, USA, 14–17 March 2019.
22. Firdaus, A.; Anuar, N.B.; Ab Razak, M.F.; Sangaiah, A.K. Bio-inspired computational paradigm for feature investigation and malware detection: Interactive analytics. *Multimed. Tools Appl.* **2017**, *77*, 17519–17555. [CrossRef]
23. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C.E.R.T. Drebin: Effective and explainable detection of android malware in your pocket. In Proceedings of the Network and Distributed System Security Symposium (NDSS) Symposium 2014, San Diego, CA, USA, 23–26 February 2014; Volume 14, pp. 23–26.
24. Altaher, A. An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and permission-based features. *Neural Comput. Appl.* **2016**, *28*, 4147–4157. [CrossRef]
25. Abdulla, S.; Altaher, A. Intelligent approach for android malware detection. *KSII Trans. Internet Inf. Syst.* **2015**, *9*, 2964–2983.
26. Altaher, A.; Barukab, O. Android malware classification based on ANFIS with fuzzy c-means clustering using significant application permissions. *Turk. J. Electr. Eng. Comput. Sci.* **2017**, *25*, 2232–2242. [CrossRef]
27. Garg, S.; Baliyan, N. A novel parallel classifier scheme for vulnerability detection in android. *Comput. Electr. Eng.* **2019**, *77*, 12–26. [CrossRef]
28. Mori, T. Information Gain Ratio as Term Weight: The Case of Summarization of Ir Results. In Proceedings of the COLING 2002, the 19th International Conference on Computational Linguistics, Taipei, Taiwan, 26–30 August 2002.
29. Belgiu, M.; Drăguţ, L. Random forest in remote sensing: A review of applications and future directions. *ISPRS J. Photogramm. Remote Sens.* **2016**, *114*, 24–31. [CrossRef]
30. Friedrich, T.; Kötzing, T.; Krejca, M.S.; Sutton, A.M. The Compact Genetic Algorithm is Efficient under Extreme Gaussian Noise. *IEEE Trans. Evol. Comput.* **2017**, *21*, 477–490. [CrossRef]
31. Itano, F.; de Abreu de Sousa, M.A.; Del-Moral-Hernandez, E. Extending MLP ANN hyper-parameters Optimization by using Genetic Algorithm. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 8–13 July 2018.
32. Moayedi, H.; Raftari, M.; Sharifi, A.; Jusoh, W.A.W.; Rashid, A.S.A. Optimization of ANFIS with GA and PSO Estimating α Ratio in Driven Piles. *Eng. Comput.* **2020**, *36*, 227–238. [CrossRef]
33. Fu, G.H.; Yi, L.Z.; Pan, J. Tuning model parameters in classimbalanced learning with precision-recall curve. *Biom. J.* **2019**, *61*, 652–664. [CrossRef]
34. Davis, J.; Goadrich, M. The relationship between precision recall and ROC curves. In Proceedings of the 23rd International Conference on Machine Learning, New York, NY, USA, 25–29 June 2006.
35. Huang, T.H.-D.; Kao, H.-Y. R2-D2: ColoR-inspired convolutional neural network (CNN)-based Android malware detections. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018.
36. Singh, J.; Thakur, D.; Ali, F.; Gera, T.; Kwak, K.S. Deep feature extraction and classification of Android malware images. *Sensors* **2020**, *20*, 7013. [CrossRef]