Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 4893390, 8 pages
https://doi.org/10.1155/2022/4893390

Research Article

A Comparative Study on Detection of Malware and Benign on the Internet Using Machine Learning Classifiers

J. Pavithra and S. Selvakumara Samy

Department of Computational Intelligence, SRM Institute of Science and Technology, Kattankulathur, India

Correspondence should be addressed to S. Selvakumara Samy; kumaraselva67@gmail.com

Received 28 April 2022; Revised 27 May 2022; Accepted 2 June 2022; Published 27 June 2022

Academic Editor: Punit Gupta

The exponential growth in network usage has opened the way for people who use the Internet to be exploited. A phishing attack is the most effective way to obtain sensitive information about a target individual without their knowledge over the Internet. Phishing detection has an increased false-positive rate and is inaccurate. The motivation behind the research is to analyze and classify the applications among malware or benign with less time complexity. The main purpose is to find the algorithm which provides better accuracy for detecting the adware. The comparative analysis was made with three machine learning classifiers to find a better one. Random forest, SVM, and naïve Bayes were selected because of the better results obtained in previous research papers. Using a confusion matrix, the classifier methods were evaluated for accuracy, precision, recall, and F-measure with positive rates of both true and false. This research indicates that there are a number of classifiers that, if accurately detected, offer better reliable phishing detection outcomes. Random forest has proven to be an effective classifier with 0.9947 accuracy and a 0.017 false-positive rate. In this study, the comparative analysis reveals that the best ML classifiers have a lesser prediction accuracy for spoofing threat identification, implying that nonphishing programmers can use the best ML classifiers to evaluate the attributes of spoofing threat recognition and classification.

1. Introduction

Advertising is used to advertise, promote, and sell a product, a service. Marketing strategies have quickly moved to digital advertising as the Internet and recently developed Smartphones have grown in popularity, and this trend shows no indications of retarding. When you visit a website or use a mobile application, you will see digital advertisements [1, 2]. However, a wide spectrum of fraudulent activity has an impact on the process of placing advertisements on mobile networks and the Internet. Adware is software which extends above and beyond the appropriate marketing used within freeware as well as shareware. Adware is often distinct software launched at the same time as shareware or perhaps another equivalent application [2]. Even if the user does not run the apparently needed software, the adware will usually assist to enhance marketing [3]. With a greater portion of the global market share, Android has become the most popular mobile operating system. And over a billion Android smartphones have been purchased of which Google Play alone estimates 65 billion app downloads. The quantity of malware exploits the security of consumer gadgets considerably in terms of the tremendous rise of smart gadgets and accessible software devices. Mobile phones are commonly applied in several areas of human existence, especially economic, commercial, societal, professional, and academic endeavours. Because of these factors, smartphones are now more valuable than merely for making calls, but also for making a sustainable economic and traditional career [4]. Smartphones can also be used to obtain appropriate web services such as online ticketing, shopping, and web-based platforms. There are various options designed for various mobile software functionalities. With the rising deployment of digital devices for bank transfers, the sudden growth in malware attacks is becoming highly complicated, inflicting personal and financial output changes to individuals, requiring the development of efficient and reliable methods to identify this malware [5, 6]. The focus of secure communications is on methodologies for protecting devices from harmful data and programs. The following are the three security-related properties to consider while securing data:

(i) Confidentiality: it concerns with prohibiting illegal users from accessing data stored on a device.

(ii) Integrity: it enables online facts or data modification, i.e., data cannot be changed by intruders.

(iii) Availability: it restricts data retention or collection. The registered user is able to access and use the software and services.

Classification approaches were used to model and analyze the actions of Android users in order to distinguish between malware and benign apps. Analysis of the active or passive characteristics of apps is used to make classification choices. The integrity of the features, such as how particular they are, has a big impact on the prediction performance of a classifier. Android-based classification technologies are far from optimal in terms of accuracy. With the isolated static characteristics, the state-of-the-art classification yields a false-negative (FN) result rate. Data- and control-flow studies are used to extract these characteristics. Programs with hybrid features have a higher FN rate. While retaining a suitable FN and FP rate, most adaptive classification techniques produce increased false-positive (FP) rates. The false-positive rate [7, 8] shows the probability of benign apps that are incorrectly labelled as malicious. The main aim of this study is to analyze the legitimate keywords, classify the malware, benign adware, and find algorithms with better accuracy.

2. Related Work

Classification techniques are extensively used since they have shown they can acquire from samples and then work effectively with manual input to examine a variety of adware concerns over time. Attacks against automated algorithms, on the other hand, are conceivable because an intruder can alter inputs in order to force the algorithm to get the desired result. The researchers looked into a supervised learning method for detecting Android adware based on static and complex data [1]. The manifest file was used to gather static features, while the network trace was used to collect dynamic characteristics. By addressing these parameters, each adware sample was then grouped into a specific family after being identified as adware or benign. The researchers used neural models, random forests, AdaBoost, and SVM, among other machine learning techniques. The binary detection challenge was shown to be more difficult than the multiclass adware classification problem [2].

LagoDroid [6] is a development tool that takes a malicious payload as input and a destination category as output and then changes the feature samples to classify it as adhering with the category while keeping the sample's core concepts.

Despite Google's efforts to restrict the spread of Android malware in its official store, the app store, then the problem persists.

In Android apps, dynamic and static analytics are prominent approaches for detecting harmful apps. Despite the fact that both analyzing methodologies have strengths and drawbacks, a unique fusion malware detection analysis called mad4a was presented to reveal certain undiscovered components of Android malware by combining the benefits of these two techniques (dynamic and static) [9]. Over the last few years [10], Android operating systems are becoming increasingly popular, and the platform has made a name for itself in both mobile and Internet of Things (IoT) platforms.

However, it has become the target of malicious apps, and this demand has evolved with more security difficulties [11]. MalDozer, an automated Android malware analysis and family attribution framework that depends on orders of classification using deep learning classifiers, was suggested in [12–16]. An Android malware detection system (APK Auditor) is based on permissions that employ static analysis to analyze and classify Android applications as benign or malicious. MalDozer extracts and learns the malicious and benign behaviours from the actual sample to identify malware from the raw sequence of the app's API requests, which is presented in [17]. It may be used as a detector not just on servers, but also on smartphones and Edge computing. To protect the systems and users' security and privacy, the authors in [18] built a system that might facilitate both the analysis and removal of harmful apps. This was done by looking at the different permissions requested by an app after installation. Clustering and classification algorithms are used in this investigation. The aim of the system was to identify and eliminate malware from users' Android mobile devices. Many applications hide individual activities by encrypting them and transmitting sensitive data across the network. Because of the flexibility and transparency of Android operating systems, new surface attacks are being created every day. The author addresses the detection of two malware attacks: intent-based hijacking and authenticated session hijacking. In order to detect these two virus problems, the honey pot approach was applied. Multiple apps and their connections only with honey pot were updated and tested using a virtual machine or perhaps an Internet browser to get the desired outcome. It must be believed that runtime methodologies, rather than traditional machine learning methods, can detect hijacked malware with great precision. Mobile attackers are creating a big number of harmful apps, especially for Android because it is the most widely used mobile operating system, by changing existing apps, which often results in malware being organized into families. Applications of the same family will act in a similar manner, gathering data and transferring them to a remote server controlled by the attacker. A deep learning method can discriminate between benign and malicious Android applications. This APK analyzer is made up of three parts:

(i) A signature database is used to hold information about applications and analysis results that have been extracted.

(ii) An Android client for granting application analysis requests from end-users.

(iii) A central server that oversees the whole analysis process and communicates both with the data repository and the Smartphone app.

Many applications were gathered in order to analyze, evaluate, and construct the system. The result shows that APK analyzer can locate and highlight the majority of well-known malware. Malware detection in Android was designed to prevent malicious software from being installed on the victim's device by detecting unusual boot sequences. It assists in the identification of malware designed for the Android operating system. There was a proposal for a sequence of pattern recognition procedures with the processing levels, as follows:

(i) Analyzing

(ii) Reviewing

(iii) Making a decision

In the initial stage, the infected app runs in a secure and protected domain to retrieve a series of system calls. It generates the measurements required for decision-making during the analysis step. To begin, the Wilcoxon signed-rank test and the global alignment technique were used to compare the similarity of legal and malignant network call patterns and then determine whether the differences between samples and shared activity are substantial [19]. In [20], the authors established a convolution neural network-based technique that was applied to the probability of system calls using dynamic analysis. A dynamic analysis methodology, unlike existing machine learning-based malware detection identification methodologies [16], authorized user trust guarantees in malware detection that are unaffected by data bias and hold for both malware and legitimate classes independently. The output based on empirical evidence accuracy, usefulness, and objectivity is demonstrated using the curved prediction technique in combination with the random forest classifier.

A machine learning-based malware defensive model for IoT environment is presented in, which uses two different approaches in the selection of adversarial samples and by computing distance from cluster centres and probability values, the method performs classification. Towards identifying the abuse in crypto mining generated by malwares, an efficient approach is presented in [20]. The method finds a set of network flow which is relevant and able to classify a set of flow generated by crypto mining. The method works according to the network flow and finds malicious flow generated.

The Android particular intent explicit and implicit features have produced a faster and more effective outcome than the Android permission function for identifying assaults [21]. In comparison to other well-known and explored qualities such as authorization, the intent is a semantic-based feature capable of storing malware's intents. This has been used as a backdoor to access numerous transmissions in the Android framework. This interaction is frequently driven by an intent messaging object on the Android platform, which is a late runtime legally enforceable communicating object.

3. Methodology

There are three steps were involved to carry over the research: Dataset Preparation, Machine learning techniques are used in a variety of situations, and their performance is evaluated. In Figure 1, the architecture framework contains the preprocessed dataset and it should be loaded at first. In the feature selection, 80 features were taken for analysis. Among the 80 features, adware-related features are selected for the classification. 70% of data is taken for training and 30% is for testing. The model was run with the help of selected three machine learning classifiers. The test results will be evaluated, and a comparison will be made for analyzing a better algorithm based on higher accuracy. Then, the false-positive and true-positive rates will be calculated by a confusion matrix.

3.1. Dataset Preparation. The dataset collection consists of APK samples compensate the (Adware-A). In 631956 samples from the Android malware adware, the dataset employed 80 characteristics of various variable kinds. For raw analysis of data, the dataset is transformed to .csv format (a machine learning-compatible format). The dataset contains 80% samples for training and 20% samples for testing the applications, and the public raw dataset was taken for the reference to compare the results [22–24].

3.2. Application of Algorithm. Among several algorithms, many researchers suggested and attained higher accuracy results in these three algorithms such as random forest, naïve Bayes, and SVM were applied in this study to properly categorize the dataset, as well as a 5, 10, and 70% for splitting cross-validation. The results of extensive testing on multiple datasets, with diverse learning processes, led to the selection of 10, 5, and 70 cross-validations, However, 10-fold has shown to be the ideal amount of folds for obtaining the most accurate measurement. A specified number of folds is chosen for cross-validation, as well as the data are arbitrarily segmented into 10 parts, and each of which roughly reflects the class in the same ratio as in the entire dataset. Before being applied to the remaining set, the constructivist approach is trained on the remaining nine-tenths of each partition. As a result, the learning method is repeated ten times across different training sets. Eventually, the total percentage of the 10 error predictions was

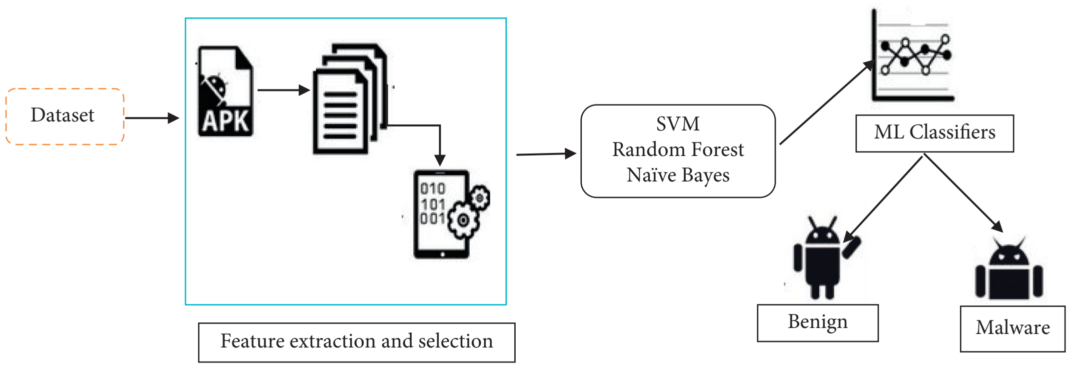

Figure 1: Architecture framework.

combined together to provide an overall error estimate, and the 5-fold and 2-fold error estimates were calculated in the same way. For comparison, the dataset had also been analyzed using a proportion, which allows you to test a specific percentage of the data; in this example, a 70% split was employed.

3.3. Feature Extraction. In order to set up an efficient Android adware categorization, it is recommended to acquire resilient and most relevant elements such as user permissions, user-specific, flow, count features and receiver's information, intent filters, process name, and files underneath analysis software. All of the above features are retrieved through reverse-engineering, which breaks down the feature APK into simple code, which is then modified and transformed back into an APK file. A simple APK disassembler is used to reverse engineer the apps.

All activities, services, broadcast receivers, and content providers are included in the app's components. Basic properties, including the name of the Kotlin or Java class, must be defined for each component. It can also declare capabilities like which device configurations it can support and intent filters that describe how the component can be launched. The name also serves as the application ID, which must be unique globally in order for your app to be published on Google Play. The build tools, however, override the package name at the conclusion of the build process by leveraging the applicationId field from the build. Gradle file (used by Android Studio projects). This will not be an issue as long as the package name in the manifest matches the applicationId in the build file. If these two values differ, see how to configure the application ID to learn about the differences between "package name" and "application ID."

Features including constant strings from binaries, permissions, carriers, transmitters, intent filters, and process names were extracted from Android manifest.xml files by reverse-engineering the Android platform. After reverse-engineering an app, constant strings actually take the binaries of Android apps that exist in a folder. Attackers can take advantage of a device by reversing the application and altering the persistent strings [25–27]. After obtaining the source code of an application through reverse-engineering, hackers alter constant strings, repackage the application, and submit it to play stores. Adware attackers can modify the constant string, for example, [constant-string v1, "Rooted Device"] and [constant-string v1, "Device Not Rooted"]. Then, in parallel, by altering constant strings in Android applications, they can launch several attacks. Android provides significant advantages to reverse engineer that iOS does not. Because Android is open source, you can examine its source code at the Android Open Source Project (AOSP) and update the operating system and its standard tools in any way you wish. Even on normal retail devices, activating developer mode and side loading programs can be done without jumping through too many hoops. Here, Algorithm 1, extracts the data from the input dataset and performs the PCA analysis, checks with the threshold value, and performs the ranking process based on the mean value. There are a variety of conveniences to make your life easier, from the strong tools that come with the SDK to the large range of available reverse-engineering tools.

Every Android app has a single manifest.xml file that requests all user permissions. Users will not be able to install the app until they accept all of the permissions that have been requested. Keywords (manifest feature) are taken from applications in addition to constant strings. As stated in Table 1, a list of keywords is produced for this purpose based on information from current malware and legitimate applications. Table 1 contains keywords gathered from malicious manifest.xml files of Android applications. The keywords were gathered to assist in the detection of dangerous software. In the manifest.xml file, each term in Table 1 has a specific function, such as "Read SMS," which reads sent and received messages from an Android phone user. In Android malicious apps, the terms Read MSG, Send MSG, Duration, and a few more listed in Table 1 were commonly used. We have extracted all the existed keywords frequently in manifest files of malicious software by analyzing harmful applications. The keyword structure is compared to genuine and malicious manifest.xml files once the keywords have been extracted. Here, the RRT represents the Rapidly exploring random trees, flowpkts denotes flow packets, and avgpkts denotes average packets.

```
(i) Steps: Input ⟶Dataset (Malware and benign)
(ii) **Output.** Feature extraction (Ranked Features)
(iii) Extraction ⟶Through PCA (Mean and threshold value based on 100 features)
      ⟶Classes.dex ⟶ android manifest.xml (AXML jar) ⟶ outputextraction If
(iv) Extraction $E$ = extracted features
(v) Apply ⟶bakenal tool – .smalicious files, API functions and permissions
(vi) THEN
(vii) Vectorization–matrix values split together
(viii) Feature vector ∊∊feature extracte values ∀ columns,
(ix) Extracted features ----through vector (6features (Ranking based on threshold PCA))
(x) Features ranked (for classification)
(xi) END
```

Algorithm 1: Feature Extraction.

Table 1: Android application keyword.

List of keywords taken as features				
Permissions	Count	Section-flow	Intent-filter (action)	General specific
Read_MSG	std_fpktl	Total_fhlen	downUpRatio	bAvgBytesPerBulk
Send_MSG	std_bpktl	total_bhlen	fAvgSegmentSize	bAvgPacketsPerBulk
Duration	total_fiat	fPktsPerSecond	fHeaderBytes	bAvgBulkRate
total_f packets	Fpsh_ cnt	bPktsPerSecond	fAvgPacketsPerBulk	bAvgSegmentSize
max_fpktl	bpsh_cnt	flowPktsPerSecond	fAvgPacketsPerBulk	RRT_samples_clnt
max_bpktl	furg_cnt	flowBytesPerSecond	fAvgBulkRate	Act_data_pkt_forward
Mean_ fpktl	Min_ fpkt	—	avgPacket size	—
mean_bpkt	Max_fpkt	—	—	—

The process of supervised classification is to classify benign and malware applications in the dataset. As a result, analyzing all malicious application manifest.xml files in our data set can create a data structure. The full keyword list used in this classification is shown in Table 1. As Table 1 describes, the keywords over previously developed classification were the improved ones.

3.4. *Classification of Malware and Benign.*

Before extracting features from adware and benign apps, a data repository of applications and Android systems was gathered. Binaries files and manifest.xml files were among the features extracted. Here, Algorithm 2 provides the experimental details about the algorithm analysis based on the feature ranks and classifies the APK whether it is benign or malware. In Algorithm 2, Fn refers to the features, and A is variable which is assigned for the algorithms. Keywords include intent filters (action), broadcast receivers, senders, permissions, and process names. Whereas constant strings, application programming interface calls, and system calls are made with built-in binaries. Constant strings from both benign and malicious applications were collected at random. Android manifest.xml files of applications from Android were collected for analyzing malware samples in the case of keywords (manifest features).

3.5. *Evaluation Metrics.*

The following evaluation metric refers to the performance metrics:

$$\textbf{Accuracy} = \frac{\textbf{TP} + \textbf{TN}}{\textbf{TP} + \textbf{TF} + \textbf{TN} + \textbf{FN}}. \tag{1}$$

For random forest, the accuracy rate is 0.9947.

Equation (1) denotes that true positive (TP) refers to the number of right malware classification predictions, whereas false negative (FN) refers to the number of incorrectly classified malware occurrences in the set. TPR stands for recall and sensitivity, among other things:

$$\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FP}}. \tag{2}$$

Equation (2) refers that the precision score and false positive (FP) are the number of misclassified predictions of benign apps:

$$\textbf{F} - \textbf{Measure} = \frac{2 * \textbf{Precision}}{\textbf{precision} + \textbf{Recall}}. \tag{3}$$

Equation (3) indicates that the entire dataset was subjected to a 5,10-fold cross-validation, and 70-percentage split into training data. For the performance analysis, comparing the score of parameters such as accuracy, precision, recall, and F-measure were employed, as summarized in the result.

```
(i) Steps:
(ii) **Input:** 1. Features (F1,F2,F3.....Fn)
(2) Algorithms: A (SVM, Naive Bayes, random forest)
(iii) **Output** Classification of.apk (benign or malware
(iv) Begin
(v) Extract the F1- Fn features
(vi) Select data points ⟶features (ranked)
(vii) Apply on algorithms (here referred SVM, Naive Bayes, random forest)
(viii)       For $i = 1$ to $n$ do//$i$ represents the features.
(ix)     If the rank $r$ in Fn (equal or max similarity)
(x) Then
(xi) Classify ⟶malware or benign
(xii) Else
(xiii) Repeat the rank similarity
(xiv) Check ∀.APK
(xv) Apply on Time Complexity
(xvi) Find the suitable algorithm based on Time Complexity.
(xvii) END
```

Algorithm 2: Classification of APK.

Table 2: Comparison of techniques for the parameters.

Techniques	Precision	F-measure	Recall
SVM	0.871	0.817	0.833
Random forest	**0.9947**	0.993	0.99
Naïve Bayes	0.784	0.718	0.789

4. Result Analysis

The dataset was reduced from 631956 individuals and 80 attributes to 3799 individuals and 60 attributes, which were employed in the study using a 70% split, and 5- and 10-fold cross-validation. This table shows the precision, accuracy, and reliability of the results. Table 2 depicts the outcomes in terms of accuracy, precision, recall, and F-measure. The obtained datasets of both bloatware and whiteware are used in experiments.

4.1. Precision, F-Measure, and Recall. Precision refers to the percentage of important recollected instances, whereas recall refers to the percentage of important recollected instances. Precision and recall are both dependent on a thorough understanding of and assessment of significance. Discussing the accuracy and recall scores in Figure 2 explains the results with one measure are compared to a specified level on the other measure, or the two measures are combined into one. Because both precision and recall are required to be higher, a high F-measure is necessary, and random forest has the greatest precision value of 0.994.

Here, different parameters such as precision, F-measure, and recall with the three algorithms explain the detection accuracy which is obtained on the experimental results. Figure 2 compares the algorithms for precision parameter which shows that the random forest classifier has better results. When compared with F-measure, the same random forest has provided the higher results shown in Figures 3 and 4.

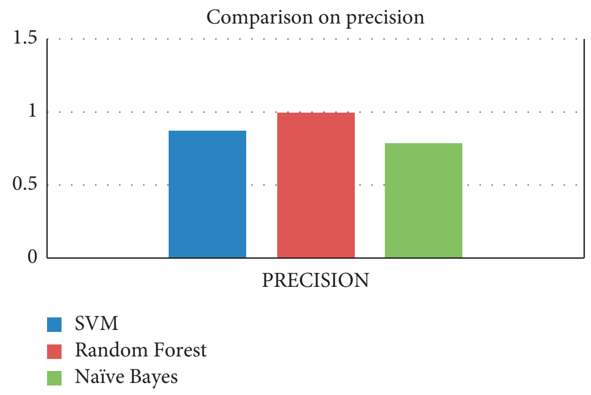

Figure 2: Comparison on precision.

4.2. Accuracy. Accuracy is a metric that indicates how accurate a prediction is. It does not take positives and negatives into account separately.

The other performance measurements are used in addition to accuracy that is mentioned in Figure 5. Random forest has a value of 0.9947, SVM has a value of 0.9625, and naive Bayes has a value of 0.9412 in this study.

4.3. Time Complexity. Time complexity measures the value of time taken for classification of various algorithms based on the given samples. It has been measured as follows:

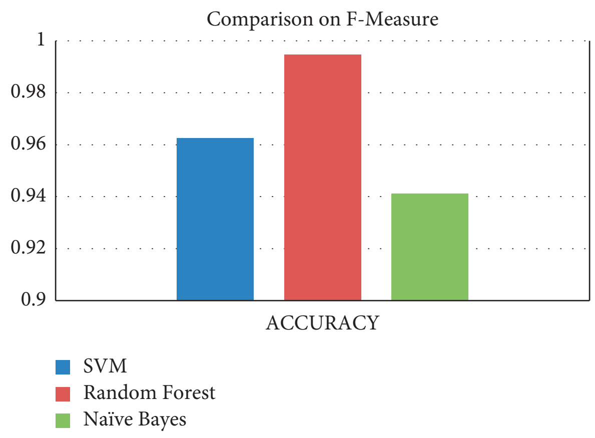

Figure 3: Comparison of F-measure.

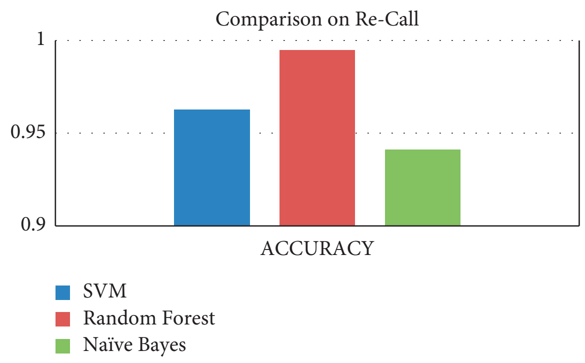

Figure 4: Comparison of recall.

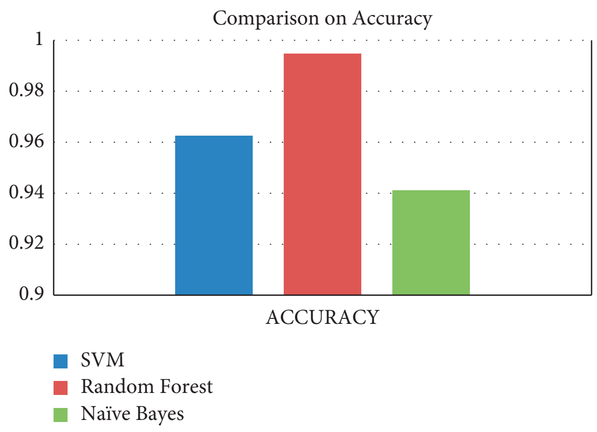

Figure 5: Comparison of accuracy.

$$\text{TimeComplexity} = \frac{\text{TotalTimeTakenforMalwareClassification}}{\text{TotalNumberofTestsamplessubmitted}}. \quad (4)$$

With reference to equation (4), the analysis is performed by considering different number of techniques and features and the time taken for prediction. The processing time for each technique has been mentioned in Table 3. In each case, it compares the results with the existing approaches.

Table 3: Time complexity for the 631956 samples.

Techniques	Time complexity
631956 samples	
SVM	1 min
Random forest	94 seconds
Naïve Bayes	1 minute 20 seconds

5. Conclusion

Using a machine learning classification approach, this paper developed an efficient and reliable adware classification. Variations in parameters and their impact on classification accuracy have been investigated. Random forest outperformed other classification algorithms out of three (3) distinct classification methods utilized in an experiment on an adware dataset with two test choices. According to the obtained results, random forest performed exceptionally well for adware classification with an accuracy of 0.9947, outperforming other well-known classification algorithms such as naive Bayes and SVM. On the larger dataset, the time complexity for the random forest is very less when compared to other techniques. It shows that it analyzes and classifies the application whether it is malicious or benign. As a result, it is recommended that more experiments should be conducted on a larger number of adware datasets from various sources, with future research focusing on new methodologies that will be applied to a larger number of classification algorithms for adware dataset analysis. Experiments show that the strategy outperforms other similar strategies. The next aim is to work on additional variation features in parallel to improve classification accuracy and eliminate false positives.

6. Future Work

Malware has a long history of posing a severe security danger to computers. The rapid development of antidetection technology has limited the capability of traditional detection approaches based on static and dynamic analysis. However, due to viral differences, feature extraction is difficult, making traditional neural network applications inefficient. Extended versions of PCA and XGBoost were used to perform the extraction and classification process. The results are projected to improve in accuracy in the future.

Data Availability

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy concerns.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] S. Suresh, F. Di Troia, K. Potika, and M. Stamp, "An analysis of Android adware," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 3, pp. 147–160, 2019.

[2] S. Yilmaz and S. Zavrak, "Adware: a review," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 6, pp. 5599–5604, 2015.

[3] L. A. Freeman and A. Urbaczewski, "Why do people hate spyware?" *Communications of the ACM*, vol. 48, no. 8, pp. 50–53, 2005.

[4] S. Shefali, R. Jolivot, and W. Choensawat, "Android malware classification based on mobile security framework," *IAENG International Journal of Computer Science*, vol. 45, no. 4, pp. 514–522, 2018.

[5] A. Altaher and O. Mohammed, "Intelligent hybrid approach for Android malware detection based on permissions and API calls," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 6, pp. 60–67, 2017.

[6] J. Y. Ndagi and J. K. Alhassan, "Machine Learning Classification Algorithms for Adware in Android Devices: A Comparative Evaluation and Analysis," in *Proceedings of the In2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*, pp. 1–6, IEEE, Abuja, Nigeria, March 2019.

[7] S. Yerima, "Android Malware Dataset for Machine Learning 2 figshare. Dataset," 2018.

[8] M. Alazab, R. Layton, S. Venkataraman, and P. Watters, "Malware detection based on structural and behavioral features of API calls," 2010.

[9] K. Vijayan, G. Ramprabu, S. Selvakumara Samy, and M. Rajeswari, "Cascading model in underwater wireless sensors using routing policy for state transitions," *Microprocessors and Microsystems*, vol. 79, Article ID 103298, 2020.

[10] A. T. Kabakus, T. Abdullah, and I. AlperDogru, "An in-depth analysis of Android malware using hybrid techniques," *Digital Investigation*, vol. 24, pp. 25–33, 2018.

[11] F. Tchakounte, *A Malware Detection System for Android*, Universität Bremen, Bremen, Germany, PhD diss, 2015.

[12] C. Arun, S. Karthick, S. Selvakumarasamy, and S. Joseph James, "Car parking location tracking, routing and occupancy monitoring system using cloud infrastructure," *Materials Today Proceedings*, 2021.

[13] K. O. Elish, D. D. Y. XiaokuiShu, B. G. Ryder, X. Jiang, and X. Jiang, "Profiling user-trigger dependence for Android malware detection," *Computers & Security*, vol. 49, pp. 255–273, 2015.

[14] O. Mirzaei, J. de Fuentes, J. Tapiador, and L. Gonzalez-Manzano, "AndrODet: an adaptive Android obfuscation detector," *Future Generation Computer Systems*, vol. 90, pp. 240–261, 2019.

[15] M. Rhode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Computers & Security*, vol. 77, pp. 578–594, 2018.

[16] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: permission-based Android malware detection system," *Digital Investigation*, vol. 13, pp. 1–14, 2015.

[17] R. Ahuja, V. Maheshwari, R. A. AbihaKazmi, A. Gupta, R. Arora, and A. Gupta, Malicious apps identification in android devices using machine learning algorithms," *International Journal of Sensors, Wireless Communications & Control*, vol. 10, no. 4, pp. 559–569, 2020.

[18] A. A. Ali and A. S. H. Abdul-Qawy, "Static analysis of malware in android-based platforms: a progress study," *International Journal of Computing and Digital Systems*, vol. 10, pp. 321–331, 2021.

[19] V. G. Shankar and G. Somani, "Anti-Hijack: runtime detection of malware initiated hijacking in android," *Procedia Computer Science*, vol. 78, pp. 587–594, 2016.

[20] F. Martinelli, F. Marulli, and F. Mercaldo, "Evaluating convolutional neural network for effective mobile malware detection," *Procedia Computer Science*, vol. 112, pp. 2372–2381, 2017.

[21] M. Almseidin, S. K. MaenAlzubi, and A. Mouhammd, "Evaluation of machine learning algorithms for intrusion detection system," in *Proceedings of the InIEEE 15th International Symposium On Intelligent Systems And Informatics (SISY)*, pp. 000277–000282, IEEE, Subotica, Serbia, October 2017.

[22] K. Raman, "Selecting features to classify malware," 2012.

[23] M. Al-Kasassbeh, S. Mohammed, M. Alauthman, and A. Almomani, "Feature selection using a machine learning to classify a malware," in *Handbook of Computer Networks and Cyber Security*, pp. 889–904, Springer, Berlin, Germany, 2020.

[24] A. Altyeb, A. ALmomani, M. Anbar, and S. Ramadass, "Malware detection based on evolving clustering method for classification," *Scientific Research and Essays*, vol. 7, no. 22, pp. 2031–2036, 2020.

[25] A. Feizollah, R. S. NorBadrulAnuar, R. Salleh, W. Ainuddin, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[26] W. Z. ZarniAung, "Permission-based android malware detection," *International Journal of Scientific & Technology Research*, vol. 2, no. 3, pp. 228–234, 2013.

[27] S. SelvakumaraSamy, V. Sivakumar, T. Sood, and Y. S. Negi, "Intelligent web-history based on a hybrid clustering algorithm for future-internet systems," in *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, pp. 571–581, Springer, Berlin, Germany, 2020.