ITU Journal of Wireless Communications and Cybersecurity

ITU/JWCC

Ransomware, Spyware, and Trojan Malware Detection for Android Using Machine Learning

Swati Shilaskar[1], **Shripad Bhatlawande**[1], **Akhil Bhalgat**[1, 2], **and Niranjan Bharate**[1, 3]

[1] Department of Electronics and Telecommunication Engineering Vishwakarma Institute of Technology, Pune, 411037, India
[2] School of Management, University of Texas, Dallas, 75080, USA
[3] Devtech Consulting, Pune, 411015, India

Abstract: The threat posed by malware has increased with the growth of technology. This makes malware detection a crucial problem. It specifically pertains to the heightened security risks that the underlying programs and their users frequently encounter. On the CIC-MalMem2022 dataset, experiments were executed. KNN, Decision Tree, Random Forest, GaussianNB, and AdaBoost were used for binary classification and multiclass classification. Additionally, the effectiveness of the employed algorithms has been evaluated. The machine learning models were optimized by tuning the hyperparameters. Random Forest and AdaBoost both achieved binary classification accuracy of 99.99%. Optuna Hyperparameter tuning for Random forest based multiclass classification performed with an accuracy of 88.31%.

Keywords: Android, malware detection, CICMalmem-22, machine learning, Optuna.

Makine Öğrenimini Kullanarak Android için Fidye Yazılımı, Casus Yazılım ve Truva Atı Kötü Amaçlı Yazılım Tespiti

Özet: Teknolojinin gelişmesiyle birlikte kötü amaçlı yazılımların oluşturduğu tehdidin de artış göstermesi kötü amaçlı yazılım tespitini önemli bir sorun haline getirmektedir. Bu da özellikle temel programların ve kullanıcılarının sıklıkla karşılaştığı yüksek güvenlik riskleriyle ilgilidir. CIC-MalMem2022 veri setinde deneyler gerçekleştirildi. İkili sınıflandırma ve çok sınıflı sınıflandırma için KNN, Karar Ağacı, Rastgele Orman, GaussianNB ve AdaBoost kullanıldı. Ayrıca kullanılan algoritmaların etkinliği de değerlendirilmiştir. Makine öğrenimi modelleri, hyperparametreler ayarlanarak optimize edildi. Random Forest ve AdaBoost'un her ikisi de %99,99'luk ikili sınıflandırma doğruluğuna ulaştı. Rastgele orman tabanlı çok sınıflı sınıflandırma için Optuna Hiperparametre ayarı %88,31 doğrulukla gerçekleştirildi.

Anahtar Kelimeler: Android, kötü amaçlı yazılım tespiti, CICMalmem-22, makine öğrenme, Optuna.

RESEARCH PAPER

Corresponding Author: Swati Shilaskar, swati.shilaskar@vit.edu

Reference: S. Shilaskar, S. Bhatlawande, A. Bhalgat, and N. Bharate, (2024), "Ransomware, Spyware, and Trojan Malware Detection for Android Using Machine Learning," *ITU Journ. Wireless Comm. Cyber.*, 1, (1) 1–8.

Submission Date: Feb, 14, 2024
Acceptance Date: May, 17, 2024
Online Publishing: Sept, 30, 2024

1 INTRODUCTION

The term "malware," which is derived from "malicious software," describes any program designed to damage computer systems or steal confidential information. One of the biggest threats to computer security is malware since it may hurt both individuals and businesses. Malware becomes increasingly sophisticated as technology develops, which makes it more challenging to identify and avoid. As a result, efficient malware detection systems are now more crucial than ever. With a global market share of 71.64%, Android phones are predicted to be owned by 3.3 billion people globally, making them the most frequently used operating system [1]. Therefore, this makes Android a prime target for malware attacks. It greatly jeopardizes users' security and privacy. End-to-end technique for analyzing characteristics extracted from Android applications is described in [2] and is based on Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNNs). It automatically connects static and dynamic features to assess if a program is harmful. They had a 96.76% accuracy rate. To report current problems faced while detecting malware, [3] used the visualization of data and adversarial learning on ML-based classifiers to efficiently find various malwares groups, taking into account threats from adversarial instances and the enormous rise in malware variants. They addressed a deep feature extraction approach to analyze malware in the context of recent deep learning (DL) advancements. These features from CNN were obtained and entered into Support Vector Machine (SVM) classifier in order to categorize malware [4]. By showcasing the inadequacies and constraints of the existing malware analysis tools, Severi et al. [5] developed a new system. They wanted to gather the system's full set of traces and enable replication using the Malrec platform. Petrik et al. [5] [6] carried out Simple binary classification from raw data and device memory dumps were used to identify malware. The results indicated that it is not accurate enough to determine the various characteristics of stride length and time. The usefulness of DL algorithms in the classification of malware was examined [7]. This study evaluated the malware detection algorithms Long short-term memory (LSTM), Gated Recurrent Unit (GRU), and CNN for static and behavior-based detection. Concurrently, a combination of CNN and LSTM models was developed. With 99.31% accuracy, the suggested hybrid model surpassed the competition. In addition, a random forest classifier was created by Ahmadi, Wüchner and Mao et al. [8] [9] ([10]) to identify malware using a number of factors, namely system calls, the file system, and other features. A DL-based approach for automated Android malware classification is presented by McLaughlin et al. [11]. The solution integrates deep neural networks and static feature analysis to extract properties from the code and Android application's metadata. CNN and LSTM networks are combined to form deep neural network's architecture of the system. The system achieved 99% accuracy on a dataset of 10,000 Android applications. The study highlights the importance of feature selection and engineering in achieving high detection accuracy and demonstrates the promise of DL models for automatic detection of malware. The approach used by Vinayakumar, et al. [12] for extraction of features from the code and metadata of Android applications combines static analysis and LSTM networks. To understand malware patterns and characteristics, the LSTM network is trained on a vast sample of harmless and malicious Android applications. On the dataset of 3,824 Android applications, the system obtained an accuracy of 98.12%. The study shows that LSTM networks are successful in detecting Android malware and emphasizes the significance of feature selection and engineering in achieving high detection accuracy. The suggested approach has the potential to identify previously undiscovered and unknown malware. Analysis of the efficacy of a single extracted feature from APK files for binary classification in malware detection was performed . It was examined on a dataset of 4,992 Android applications and using the SVM, attaining a 95.1% accuracy [13] . The suggested approach has limits in identifying increasingly complicated and sophisticated malware, which may necessitate additional characteristics. In order to find patterns in system-wide data, namely the storage available and transferred packet consumption, the authors proposed a customized DL system based on prevailing models like Encoder and the ResNet model [14]. This system would be used to detect sensitive app behaviors. using a machine learning-based approach to malware classification for Android called Random Forest classification [15]. To extract features from the code and metadata of Android applications, the proposed approach utilizes a hybrid approach of static analysis and feature engineering. Carrier, T. et al. [16] propose an approach to extract properties from the memory of Android applications that requires both static and dynamic features. Then, machine learning classifier is developed using the information gathered to distinguish between benign and adverse applications, including those that have been hidden. The system's accuracy on the CiC-Malmem -22 dataset was 99.7% .

2 METHODOLOGY

The proposed method uses machine learning classifiers like Random Forest, KNN and Decision Tress to detect malicious applications. The proposed technique also classifies the malignant applications into Benign, Ransomware, Spyware and Trojans. The short flowchart of the proposed method is shown in Fig. 1.

The proposed method leverages the CiCMalmem-2022 dataset, designed specifically to address the challenge of obfuscated malware detection. Obfuscated malware refers to malicious software deliberately concealed to evade detection and removal. This dataset was meticulously cu-

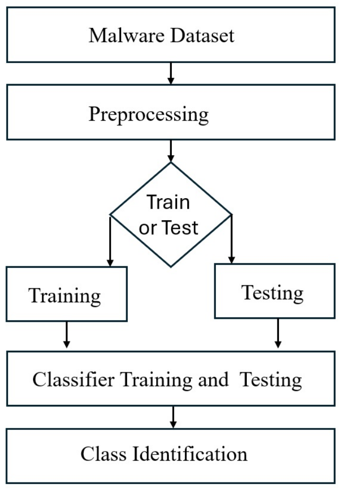

Fig. 1 Process flow of the system.

rated to emulate real-world scenarios, featuring prevalent strains of obfuscated malware encountered in actual cyber threats. This dataset comprises a total of 58,569 records. The collection of harmful and benign dumps yields, 29,298 benign and 29,298 malicious entries. Within the malicious category, the samples are further classified into three main classes: ransomware, spyware, and Trojan horses, with 9,535, 9,866, and 9,907 entries respectively. It's worth noting that while ransomware, spyware, and Trojan horses are subdivided into subclasses, these subclasses were not utilized in the current study. The dataset comprises several distinct feature sets, each offering insights into different aspects of malware behavior. These feature sets include pslist, dlllist, handles, ldrmodules, malfind, psxview, modules, and svcscan. pslist, which provides details on process listing, such as processes, parent processes, and average thread count per process, encompassing 5 features; dlllist, which encompasses details about DLL (Dynamic Link Library) usage, covering DLLs loaded by processes and the average number of DLLs per process, with 2 features; handles, offering insights into handle usage, providing information on the total number of handles and the average number of handles per process, with 9 features; ldrmodules, capturing details about loaded modules, including modules not loaded, initialized, and in memory, comprising 6 features; malfind, focused on memory scanning for malware, detecting memory injections, with 3 features; psxview, providing an overview of process views, including processes not listed, not in specific process pools, and not in specific lists, consisting of 14 features; modules, offering insights into loaded modules, with 1 feature; and svcscan, providing information on scanned services, kernel drivers, and process services, consisting of 7 features. These feature sets, each with its respective number of features, collectively contribute to the comprehensive analysis of malware behavior in the dataset. 26 Features extracted by using listed VolMemLyzer feature extractor are listed in [17]. The accuracy of 99.99% is achieved for binary classification in the pioneering work described in this paper.

For the dataset to be appropriate for categorization, some data preparation procedures are needed. These techniques are crucial for increasing the potency of classification models and transforming data into a machine-learnable format. In this study, the categorical variables In order to prepare them for machine learning classification, Benign and Malicious are given the two different values of 0 and 1, respectively. The CIC-MalMem2022 dataset, on the other hand, is a dataset that is evenly balanced with two classes: benign and malware. It shows resilience to overfitting since the dataset is evenly balanced. Additionally, we removed elements from the dataset that don't affect machine learning's effectiveness. These characteristics have zero weights and have no bearing on how the learning algorithms perform. In this research, The k-fold validation method is employed here. With this technique, the data is at random segregated into two groups: training and testing set. Then, this dataset is divided into 'k' samples, with 'k-1' samples being utilized for training and 'k' samples for testing . This entire procedure is performed k times with different training and testing datasets each time. The optimal model is then chosen based on the lowest error produced via the use of several statistical methods for error estimates. Random Forest is a supervised classification and regression ensemble learning system [18] . It entails building numerous decision trees that produce predictions using a randomly selected subset of features. During training, the technique includes randomization to reduce overfitting and make the model more resilient [19]. During prediction, the algorithm takes the input data and sends it through each decision tree, with the final prediction determined by the majority vote of all the trees [20]. Random forest is a versatile and strong algorithm that may offer a measure of feature relevance and can be utilized for a variety of purposes. A common machine learning method for classification and regression issues is decision trees. Creating a tree-like model of decisions and probable outcomes is one of them. The algorithm looks for the ideal properties to divide the data into subsets and produce decision nodes that maximize information gain or minimize impurity during training. The model can be visualized as an if-then flowchart, with every internal node showing a test on a feature, each branch denoting the test's output, and every leaf node denoting an outcome or prediction [21]. Decision trees can handle both category and numerical data and are easily interpretable. They are, however, susceptible to overfitting and can be sensitive to slight changes in the training data. AdaBoost, or Adaptive Boosting, is an ensemble learning technique that combines numerous weak classifiers to produce a powerful classifier.

It accomplishes this by iteratively training weak classifiers on different subsets of the training data, with a focus on examples that were misclassified in prior iterations. The information and experience of these weak classifiers are aggregated to generate the final classifier. AdaBoost successfully identifies complicated decision limits that precisely represent the underlying structure of the data via this iterative boosting method. The technique for regression problems locates the K data points in the training set that are closest to the input data point and assigns the most frequent class or the mean value of the K closest neighbors to those data points. Any distance metric, including Manhattan distance and Euclidean distance, can be used to determine how similar two data points are. It also offers precise end-user predictions established on the basic categorization principles of resemblance or distance [22]. The Gaussian Naive Bayes method is a probabilistic algorithm that computes the probability of various classes given input characteristics. It entails conducting calculations like median, variance, and probabilistic density projections, which can be computationally demanding, particularly when dealing with big datasets or high-dimensional feature spaces. Grid Search: Finding the set of hyperparameters that results in the best model performance necessitates constructing a grid of hyperparameter values and carefully examining all potential combinations of these values. In order to perform a grid search, we might need to specify a boundary because the parameter space for the machine learning approach could comprise spaces with actual or infinite values for some parameters. The hyperparameters that yield the best results are often chosen after the model's performance has been evaluated using a performance metric like accuracy, precision, or recall. The implementation is shown with a flowchart in Fig. 2.

Optuna Hyperparameter Tuning: It employs a rapid algorithm based on Bayesian optimization to search the hyperparameter space and discover the best set of hyperparameters for a given model. The framework operates by defining the area of search space for the hyper parameters and the optimization target in an experiment object. Then, with the purpose of minimizing the objective function, it employs a mix of Tree structured Parzen estimators (TPE) and other optimization algorithms to intelligently sample the hyperparameters and assess their performance [23]. This is shown in Fig. 3.

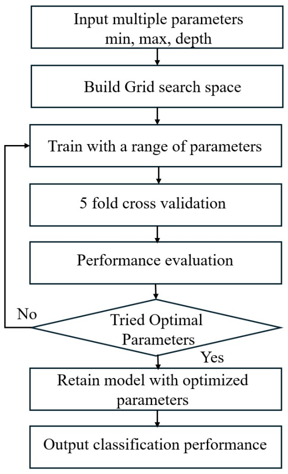
Input multiple parameters min, max, depth
Build Grid search space
Train with a range of parameters
5 fold cross validation
Performance evaluation
No
Tried Optimal Parameters
Yes
Retain model with optimized parameters
Output classification performance

Fig. 2 Flowchart of Grid Search.

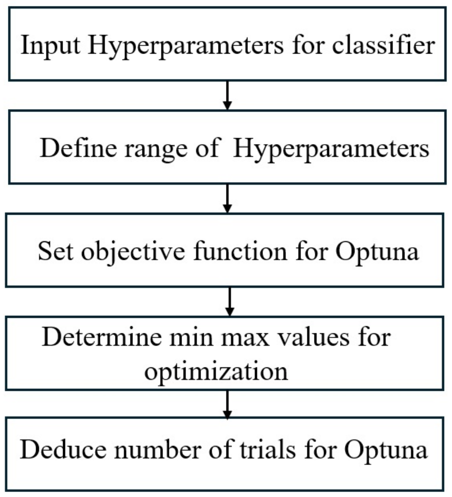
Input Hyperparameters for classifier
Define range of Hyperparameters
Set objective function for Optuna
Determine min max values for optimization
Deduce number of trials for Optuna

Fig. 3 Selection of Optuna hyperparameters.

Algorithm 1 describes hyperparamaeter selection process for Optuna optimization.

Algorithm 1 Optuna Hyperparameter Algorithm

```
1. Input dataset
2. for metric 1 to 100 do
3. Splitting dataset into 70% training set and 30% test
   set
4. Fit Min_Max_Scaler to training set.
5. for i in n_trials do
6. select new a_(n+1) by optimizing function
7. a_(n1)ārgmax Alpha(a;R_n )
8. query objective function to get b_(n+1)
9. amplify data R_(n1)
10. Compute metric for best model over test set
11. if run-time > time_limit then
12. end
```

2.1 Model Implementation and Evaluation

In this section, the proposed system uses an ensemble approach for binary classification of malware, into malignant and benign. The six classifiers that were used in the approach are mentioned in the Table 1.

Table 1 Accuracy for binary classification of malware

Classifier	Accuracy in%
Decision Tree	99.96 ± 0.02
KNN	99.92 ± 0.21
Random Forest	99.99 ± 0.005
Gaussian NB	99.25 ± 0.06
AdaBoost	99.99 ± 0.05
Logistic Regression	99.57 ± 0.17

To further classify the malignant files into their respective malware families, we also performed a multiclass classification of the CiCMalmem-22 dataset. In order to improve the parameters of the base learner and get the highest classification accuracy for Android malware applications, Hyperparameter tuning techniques such as, Grid search and Optuna were used. The correlation matrix heatmap is a graphical representation of data that is used to identify the strongest correlations between various characteristics and the target feature. Each feature in a dataset is represented by a different colour, which informs about the relationships between the features. For the demonstration, we created a heatmap of the correlation matrix, Fig. 4. As given in the Fig. 4 numbers are given to the feature names and are listed in next paragraph.

Selection of the features is then carried out to remove redundant and unnecessary features. This helps in dimensionality reduction of the data and make it more efficient. Only features with correlation value of 0.95 and above were eliminated, and the rest of were considered. Total 55 features are present in the dataset, out of which 3 are eliminated due to non variability. 52 features used to train the classifiers are given as follows: 1. pslist.nproc, 2. nppid, 3. avg_threads, 4. handlers 5. dlllist.ndlls, 6.avg_dlls_per_proc, 7. handles.nhandles, 8. avg_handles_per_proc, 9. nfile, 10. nevent, 11. ndesktop, 12. nkey, 13. nthread, 14. ndirectory, 15. nsemaphore, 16. ntimer, 17. nsection, 18. nmutant; 19. ldrmodules.not_in_load, 20. init, 21. mem, 22. load_avg, 23. init_avg, 24. mem_avg 25. malfind.ninjections, 26. commitCharge, 27. protection, 28. uniqueInjections 29. psxview.not_in_pslist, 30. eprocess_pool, 31. ethread_pool, 32. pspcid_list, 33. csrss_handles, 34. session, 35. deskthrd, 36. pslist_false_avg, 37. eprocess_pool_false_avg, 38. ethread_pool_false_avg, 39. not_in_pspcid_list_false_avg, 40. csrss_handles_false_avg, 41. session_false_avg, 42. deskthrd_false_avg 43. modules.nmodules, 44. svcscan.nservices, 45. kernel_drivers, 46. fs_drivers, 47. process_services, 48. shared_process_services, 49. nactive; 50. callbacks.ncallbacks, 51. nanonymous, 52. ngeneric. Only 32 features are selected as shown in Fig. 5. The selected feature as listed in Fig. 5 are pslist.nproc, nppid, avg_threads, avg_handlers; dlllist.ndlls, avg_dlls_per_proc; handles.ndesktop, nkey, nsemaphore, nsection, nmutant; ldrmodules.not_in_load, load_avg, malfind.ninjections, commitCharge, uniqueInjections, modules.nmodules, psxview.not_in_pslist, eprocess_pool, ethread_pool, eprocess_pool, false_avg, ethread_pool, false_avg; svcscan.nservices, kernel_drivers, fs_drivers, process_services, nactive; callbacks.ncallbacks, nanonymous, generic.

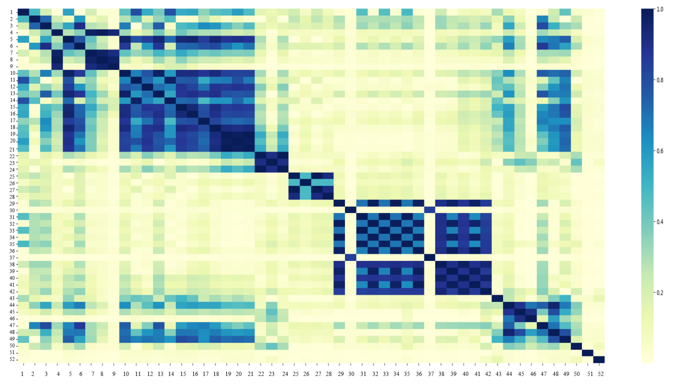
Fig. 4 Correlation heatmap of all features.

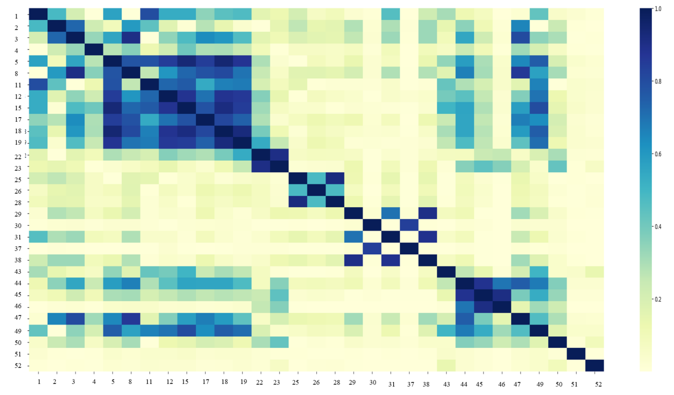
Fig. 5 Correlation heatmap of the reduced features.

In the suggested learning technique, the Random Forest algorithm was first run with the default parameters, to gauge the level of its accuracy. To further optimize the algorithm, Grid search method was employed. Fig. 6 shows the comparison of how depth of the tree impacts accuracy for Random Forest with 20, 60, and 130 trees.

As can be seen from the figure, stable accuracy is achieved at 23 depths of tree. We notice similar trends for other tree sizes. When the amount of trees in the Random

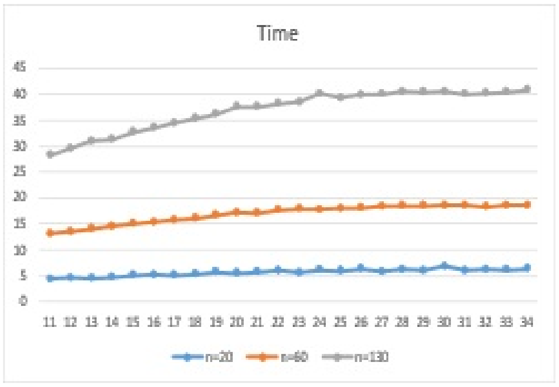

Fig. 6 Time for trees and maximum allowed depth.

Forest increases, so does its computational time. Overall trend is depicted in Fig. 7

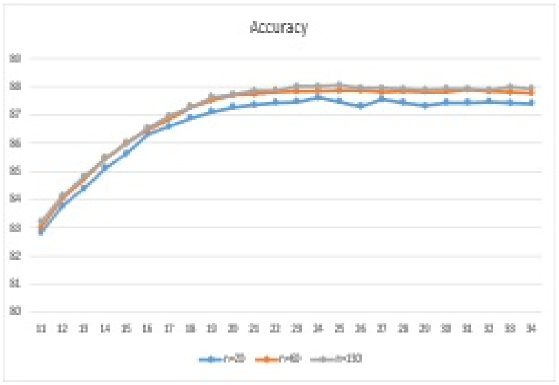

Fig. 7 Accuracy comparison with 20, 60 and 130 trees for varying depth of tree.

To optimize decision trees, varying depths were evaluated. The maximum allowed depth was 10 through 50, and their accuracy criterion was evaluated. The depth that gives the best performance was chosen, as shown in Table 2. The maximum accuracy is achieved at the depth tree of 23 with accuracy of 85.05%. It becomes stagnant as soon as the depth of trees is increased; however, the computational time also increases substantially.

By comparing the weights of numerous existing features, a case search method known as nearest neighbor determines how similar two instances are. The number of neighbors in a group of training data which lie closest to a specific value in the validation or testing set is represented by the K parameter of K-NN classifiers. K-values between 2 and 40 were examined. In Fig. 8, accuracy as K-values changed is observed.

Accuracy continues to decline as the K value of K - Nearest Neighbors increases. To improve the classification performance more samples are included to train the GaussianNB model. The datapoints that are away from the distribution mean are taken in consideration for building the model. The parameter is named var_smoothing for GaussianNB. It is plotted on x axis and accuracy on y axis to show the performance, as seen in Fig. 9.

Table 2 Accuracy mean and standard deviation performance of decision tree for varying depths

Depth	Time (s)	Accuracy (mean ± std) in %
10	1.6	80.81 ± 0.63
11	2.32	81.45 ± 0.73
12	1.83	82.22± 0.39
13	1.95	82.97± 0.20
14	1.88	83.51 ± 0.34
15	2.3	83.94 ± 0.31
16	2.1	84.18 ± 0.27
17	2.42	84.50 ± 0.33
18	2.06	84.65 ± 0.37
19	2.57	84.91 ± 0.34
20	2.32	84.89 ± 0.50
21	2.39	84.96 ± 0.25
22	2.68	85.03 ± 0.16
23	2.45	85.05 ± 0.25
24	2.65	85.02 ± 0.17
25	2.37	85.05 ± 0.13

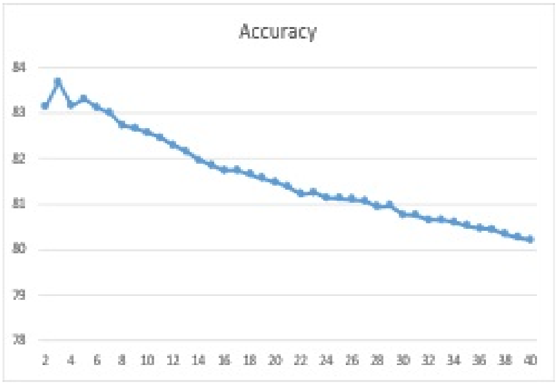

Fig. 8 Graph of accuracy with respective to varying K values.

Highest accuracy is achieved at the variance smoothing of 1.00E- 6. The accuracy starts to plateau as the va_smoothing reaches 1.00E – 10. As the n_estimators are increased its accuracy tends to decrease. The highest recorded accuracy was achieved at n_estimators = 45 seen in Fig. 10.

To further optimize the model, hyperparameters were tuned using Optuna. A cutting-edge system for automated hyperparameter optimization. Optuna provides a define-by-run user API that enables dynamic search space construction as well as effective sampling and pruning algorithms. The standard tree structured Parzen estimator (TPE) Bayesian sampling procedure was employed. An objective function is defined, that takes the accuracy score of a tree trained and evaluated with the hyperparameters

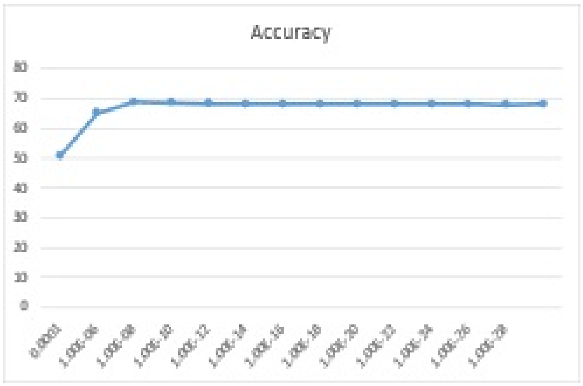

Fig. 9 Accuracy graph of GaussianNB for varying curve smoothing values.

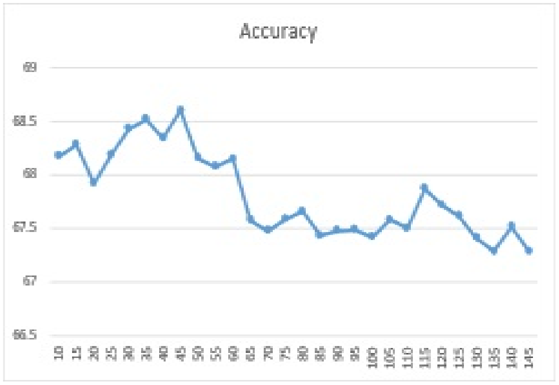

Fig. 10 Accuracy graph for AdaBoost with varying n_estimators.

provided by the trial. Integer values for max_depth and min_samples_split are set to (20,27) and (2,5) respectively. Then, by using the optimize() method TPE was tuned, and 70 trials were executed. The best hyperparameters and score are shown in Table 3.

Table 3 Multiclass classification accuracy achieved after implementation of Optuna

Classifier	Accuracy (mean ± std) in%
Decision Tree	85.26 ± 0.31
KNN	84.55 ± 0.11
Random Forest	88.31 ± 0.21
Gaussian NB	68.80 ± 0.80
AdaBoost	70.31 ± 0.57

3 CONCLUSION

Numerous categorization techniques from machine learning have been used, as detailed in this work, to determine the best method for identifying malware infections on Android devices. In this work, the potential of machine learning classifiers namely GaussianNB, AdaBoost, Logistic Regression, Random Forest, Decision Tree, and KNN is investigated to detect malware infections. The data source for the dataset was CiCMalmem-22. Random Forest and AdaBoost achieved a remarkably high accuracy of nearly 99.99%, respectively, of the samples properly identified in the trials. It is based on a 5-fold cross-validation for binary classification. This paper is also the only study that provides state of the art, multiclass classification of malware into their respective families. For multiclass classification of malware into Ransomware, Spyware and Trojan families, these classifiers were further tuned using Grid search and Optuna Hyperparameter Tuning techniques. In which, Random Forest hyperparameter tuned further by utilizing Optuna performed with highest accuracy, i.e., 88.31%. This also resulted in decreased computational times.

DECLARATION

There are no conflicts of interests to declare.

ACKNOWLEDGEMENTS

We would like to thank Vishwakarma Institute of Technology, Pune, for providing the ecosystem to work on research project.

REFERENCES

[1] admin@enterpriseappstoday.com. "Enterpriseappstoday." (2010), [Online]. Available: `https://www.enterpriseappstoday.com/stats/android-statistics.html#:~:text=There%20are%203.3%20billion%20Android,The%20latest%20version%2C%20Android%2012.0`. (accessed: 15.05.2024).

[2] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016. DOI: `10.1109/TST.2016.7399288`.

[3] X. Liu, Y. Lin, H. Li, and J. Zhang, "A novel method for malware detection on ml-based visualization technique," *Computers & Security*, vol. 89, p. 101 682, 2020.

[4] M. Asam, S. J. Hussain, M. Mohatram, *et al.*, "Detection of exceptional malware variants using deep boosted feature spaces and machine learning," *Applied Sciences*, vol. 11, no. 21, p. 10 464, 2021.

[5] M. Brengel and C. Rossow, "Memscrimper: Time-and space-efficient storage of malware sandbox memory dumps," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2018, pp. 24–45.

[6] S. S. H. Shah, A. R. Ahmad, N. Jamil, and A. u. R. Khan, "Memory forensics-based malware detection using computer vision and machine learning," *Electronics*, vol. 11, no. 16, p. 2579, 2022.

[7] H. Safa, M. Nassar, and W. A. R. Al Orabi, “Benchmarking convolutional and recurrent neural networks for malware classification,” in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, IEEE, 2019, pp. 561–566.

[8] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel feature extraction, selection and fusion for effective malware family classification,” in *Proceedings of the sixth ACM conference on data and application security and privacy*, 2016, pp. 183–194.

[9] T. Wüchner, M. Ochoa, and A. Pretschner, “Robust and effective malware detection through quantitative data flow graph metrics,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 12th International Conference, DIMVA 2015, Milan, Italy, July 9-10, 2015, Proceedings 12*, Springer, 2015, pp. 98–118.

[10] Ö. Aslan, M. Ozkan-Okay, and D. Gupta, “Intelligent behavior-based malware detection system on cloud computing environment,” *IEEE Access*, vol. 9, pp. 83 252–83 271, 2021.

[11] N. McLaughlin, J. Martinez del Rincon, B. Kang, *et al.*, “Deep android malware detection,” in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 301–308.

[12] R. Vinayakumar, K. Soman, P. Poornachandran, and S. Sachin Kumar, “Detecting android malware using long short-term memory (lstm),” *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1277–1288, 2018.

[13] D. Zhu, Y. Ma, T. Xi, and Y. Zhang, “Fsnet: Android malware detection with only one feature,” in *2019 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2019, pp. 1–6.

[14] H. Ma, J. Tian, K. Qiu, *et al.*, “Deep-learning–based app sensitive behavior surveillance for android powered cyber–physical systems,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5840–5850, 2020.

[15] M. S. Alam and S. T. Vuong, “Random forest classification for detecting android malware,” in *2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE cyber, physical and social computing*, IEEE, 2013, pp. 663–669.

[16] T. Carrier, “Detecting obfuscated malware using memory feature engineering,” 2021.

[17] T. Carrier., P. Victor., A. Tekeoglu., and A. H. Lashkari., “Detecting obfuscated malware using memory feature engineering,” in *Proceedings of the 8th International Conference on Information Systems Security and Privacy - ICISSP*, INSTICC, SciTePress, 2022, pp. 177–188, ISBN: 978-989-758-553-1. DOI: 10.5220/0010908200003120.

[18] K. M. Han J. Pei J., *Data Mining: Concepts and Techniques*. 2011.

[19] K. Alkhatib and S. Abualigah, “Predictive model for cutting customers migration from , banks: Based on machine learning classification algorithms,” in *2020 11th International Conference on Information and , Communication Systems (ICICS)*, IEEE, 2020, pp. 303–307.

[20] X. Pan, L. Zhu, Y.-X. Fan, and J. Yan, “Predicting protein–rna interaction amino acids using random forest based on submodularity subset selection,” *Computational biology and chemistry*, vol. 53, pp. 324–330, 2014.

[21] L. Rokach and O. Maimon, *Decision trees." Data mining and knowledge discovery handbook*. Springer New York, 2005.

[22] N. Ahmed, R. Ahammed, M. M. Islam, *et al.*, “Machine learning based diabetes prediction and development of smart web application,” *International Journal of Cognitive Computing in Engineering*, vol. 2, pp. 229–241, 2021.

[23] S. Shekhar, A. Bansode, and A. Salim, “A comparative study of hyper-parameter optimization tools,” in *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, IEEE, 2021, pp. 1–6.