Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2017.DOI*

# Evaluation of Machine Learning methods for Android Malware Detection using Static Features

**FERDOUS ZEAUL ISLAM[1], ASHFAQ JAMIL[2], SIFAT MOMEN[3],**

[1]Department of Electrical and Computer Engineering, North South University. Bashundhara, Dhaka, Bangladesh | (e:mail: ferdous.islam@northsouth.edu)
[2]Department of Electrical and Computer Engineering, North South University. Bashundhara, Dhaka, Bangladesh | (e-mail: ashfaq.jamil@northsouth.edu)
[3]Department of Electrical and Computer Engineering, North South University. Bashundhara, Dhaka, Bangladesh | (e-mail: sifat.momen@northsouth.edu)

This work received no sponsor or financial support

**ABSTRACT** Popularity of the Android platform has made it a prime target for security threats. Third party app stores are getting flooded with malware apps. An effective way of detecting and therefore preventing the spread of malwares is certainly necessary. Machine learning methods are being actively explored by researchers for malware detection using static and/or dynamic features extracted from android application package(apk) file. In this paper we evaluate four classifiers- Decision Tree, K-Nearest Neighbour, Linear SVM and Random Forest for detecting malware and benign android apps from static features. We applied correlation based feature selection and trained each classifier on the train set by hyper-parameter tuning with stratified 10-fold cross validation and evaluated their performance on the unseen test set. Accuracy, f1-score and area under the ROC curve (AUC) were picked as the evaluation metrics. Random Forest produced the best results, with accuracy = 0.963, f1-score = 0.947 and AUC = 0.993.

**INDEX TERMS** Android Malware detection, Machine Learning, Static Analysis of Android Malware.

## I. INTRODUCTION

In the modern world smartphones are an inseparable part of our life. According to [1] the number of smartphone users all over the world was more than 3 billion in 2019. Among different mobile operating systems, Android OS stands at top with 72.2% worldwide market share [2]. The open source operating system has paved the way to provide affordable and easy to use smartphones to people from all schools of life. However, the popularity of Android has also made it a prime target for hackers to breach security and gain access to valuable user information. A report by McAfee [3] found that there were more than 35 million android malware apps detected in 2019. Detecting new variations of malwares have become a tough ask for existing signature based methods. As a remedy, machine learning approaches are being explored widely by researchers.

In this paper, we evaluate four machine learning classifiers- Decision Tree, K-Nearest Neighbour, Linear SVM and Random Forest for detecting android malware using static features. Two datasets were obtained from [4], which were generated using two of the most renowned android malware repositories presented in- Drebin [5], Malgenome [6]. The datasets were pre-processed and merged at the start. We evaluated each classifier after applying feature selection, hyperparameter tuning and stratified 10-fold cross validation using metrics- accuracy, f1-score and area under the ROC curve.

The paper is structured into five sections. Section I(this section) is the introduction. Section II discusses related works done on android malware detection. Section III describes our research methodology- dataset acquisition, pre-processing, feature selection, and applying ML classifiers. In section IV we present the results of the ML classifiers used and compare with other notable works. Finally, section V ends with the conclusion.

## II. LITERATURE REVIEW

In this section we will provide analysis of several papers on android malware detection using machine learning. There are two approaches to a machine learning based malware detection: (i) Static Analysis and (ii) Dynamic Analysis.

### *A. STATIC ANALYSIS*

In static analysis the features from the apk file are collected before running it on device. This type of android malware detection emphasizes on less resource consumption. Some papers based on static analysis are given below,

In [4] a fusion classifier architecture was proposed. Evaluation on three different datasets showed that the proposed fusion classifier outperforms traditional ML classifiers- J48, REPTree, Random Tree and Voted Perception. The authors in [5] used a dataset covering 179 malware families with 8 sets of features and applied Linear SVM. The model predicted 94% of malwares with 1% false positive rate.

In [7] the authors evaluated performance of ML classifiers on dataset with different sets of features- API calls, API calls with parameters and permissions. KNN was applied and the API calls with parameter based feature set produced better results than the permission based one. Accuracy of 99% and 97.8% recall was reported.

In [8] class imbalance issue of malware datasets was addressed. Synthetic minority oversampling(SMOTE), random undersampling and both combined were used to counter unbalanced dataset. Additionally, Mathew Correlation Coefficient (MCC) score along with other general evaluation metrics was used. KNN, SVM and Decision Tree(ID3) were applied, among them KNN with SMOTE performed the best with accuracy 98.69%, f1-score 98.69% and MCC score 97.39%.

In [9] the authors extracted three sets of features- API calls, API call frequency, API call sequence from sample apks from a generated control flow graph of each apk. Decision Tree (C4.5), Deep Neural Network, Longest Short Term Memory were used on the 3 sets of dataset respectively. Finally, an ensemble classifier combining the three models based on soft voting on weighted sum of accuracy was built. The ensemble model produced- false positive rate 1.58%, precision 99.15%, recall 98.82% and f1-score 98.98%.

In [10] SVM with different kernels were applied to a dataset with features- API calls, parameters and return type. Linear SVM model reported best results- 99.75% accuracy, 99.52% precision and 99.54% recall.

### B. DYNAMIC ANALYSIS

In this approach the features of an apk are collected by running the it in a sand-box environment. This is a more resource consuming approach than static malware detection. Summaries of some papers based on dynamic analysis are given below.

In [11] Tong and Yan portrays an implemented hybrid Android malware detection method. This method generates malware pattern set and benign pattern set from its ability to collect execution data from both malware and benign apps. The detection accuracy of the system can reach up to an accuracy of 91.76% along with FPR less than 4% if parameters are chosen correctly. However, this method can not be used for real time malware detection because of big data processing and privacy protection on outsourcing data.

In [12] the authors proposed hardware enhanced online malware detection system which is called GuardOL. The system uses a multilayer perceptron algorithm to train a classifier and generate a set of predictions, which can classify samples with high accuracy. The system has an accuracy of greater than 90%, AUC of 0.997 and FPR of 1.2%.

In [13] the authors proposed an android malware detection system using the permissions to detect malicious software and remove it. Naive Bayes was used as the classification algorithm. However, the system is unable to detect any new family of malware.

In BRIDEMAID [14] the authors propose a malware detection system using both static and dynamic analysis. Its malware detection accuracy reached up to 99.7%. But it requires high consumption of local resources.

## III. RESEARCH METHODOLOGY

In this section we describe the entire process of our work as shown in Figure 1. At first we pre-processed and merge the collected datasets. Then we split it into test and train sets. Feature selection is applied to the train set and ML models were trained on them. Afterwards using the train set, hyper-parameter tweaking was performed with stratified 10-fold cross validation for each classifier. Finally the classification models with chosen hyper-parameter value are evaluated on the test set. The class imbalance issue of the dataset was addressed by adding evaluation metrics- precision, recall and f1-score along with accuracy for evaluation. To judge the effectiveness of a classifier, area under the ROC curve (AUC) was also considered in evaluation.

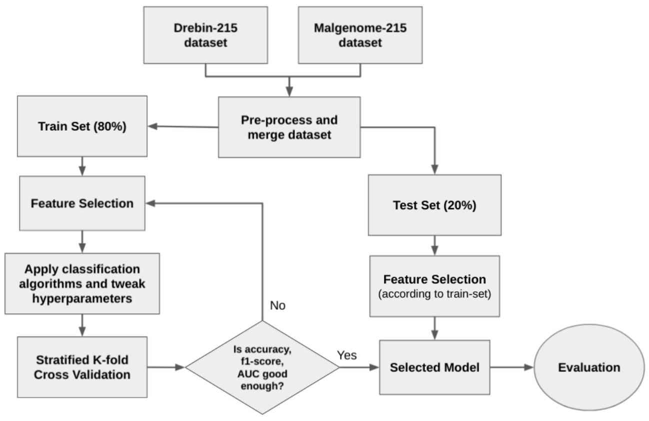
Drebin-215 dataset
Malgenome-215 dataset
Train Set (80%)
Pre-process and merge dataset
Feature Selection
Test Set (20%)
Apply classification algorithms and tweak hyperparameters
Feature Selection (according to train-set)
No
Stratified K-fold Cross Validation
Is accuracy, f1-score, AUC good enough?
Yes
Selected Model
Evaluation

FIGURE 1. Flowchart of research methodology

### A. DATASET ACQUISITION

Two datasets were acquired from [4], which were provided as a supplementary material with the paper. The malware samples of the two datasets named- Drebin-215 and Malgenome-215 originate from [5] and [6] respectively. Both the datasets contain 215 features, consisting of four feature sets: (i)API calls- method names extracted from apk's .dex file, (ii)Manifest permissions- permissions declared in apk's AndroidManifest.xml file, (iii)Intents- used for communication with other apps/services declared in the apk's AndroidManifest.xml and (iv)Command signatures- linux commands found in the apk's .dex, .jar, .so, .exe files. [7] shows that these set of features work well for machine learning. Features were extracted by [4] from Android Application Package (apk) files of the sample apps with a static analysis tool built with python. All of the features contained binary values of 1 or 0 indicating the existence or non-existence of the feature respectively. The target class contained values- 'B' denoting benign and 'S' denoting malware. Table 1 shows details of the two datasets.

| Dataset | # of instances | # of Malware instances | # of benign instances | # of features |
|---|---|---|---|---|
| Drebin-215 | 15036 | 5560 | 9476 | 215 |
| Malgenome-215 | 3799 | 1260 | 25391 | 215 |

TABLE 1. Collected datasets.

## B. PRE-PROCESSING DATA

As mentioned before we collected two datasets- Drebin-215 and Malgenome-215 from [4]. First missing values were checked on both datasets. 5 instances on the Drebin-215 dataset contained missing values for the feature- 'TelephonyManager.getSimCountryIso' which is an API call. These 5 instances were dropped. Malgenome-215 contained no missing values.

Before merging the two datasets we checked for common features on both of them. We found that the datasets had 208 features in common, the rest of the features were dropped. The target class values- 'B' denoting benign and 'S' denoting malware were converted to 0 and 1 respectively. After that, the two datasets were merged together into a single dataset- 'Drebin-Malgenome-Combined', which was used from here on. Drebin-Malgenome-Combined contains 208 features and a total 18830 instances, out of them- 12015 are malwares (63.8%) and 6815 benign (36.2%) instances. Table 2 contains details about the features of Drebin-Malgenome-Combined dataset.

| Feature Set | Description | # of features |
|---|---|---|
| API Calls | Function call extracted from .dex file | **72** |
| Manifest Permission | Permissions that an app requests, declared in the manifest file | **109** |
| Intents | Define communication method between apps/sevices, declared in AndroidManifest.xml | **23** |
| Commanded Signatures | Linux commands extracted from attached .dex, .jar, .so, .exe files etc | **4** |

TABLE 2. Features in the merged dataset- 'Drebin-Malgenome-Combined'

Finally the merged dataset was split in 80:20 ratio of train and test set. The split was made maintaining the class distribution of malware and benign instances on both sets.

## C. FEATURE SELECTION

Feature selection on the train set was done using correlation based feature selection (CfsSubsetEval) in Weka [15]. The CfsSubsetEval evaluates the worth of a subset of features by individual predictive ability of each feature as well as the degree of redundancy between them [16]. Out of the 208 features of the train set, 27 were selected. The selected features are shown in Table 3.

| Feature set | # of features |
|---|---|
| API Calls | 14 |
| Manifest Permission | 11 |
| Intents | 2 |

TABLE 3. Selected features using CfsSubsetEval in Weka.

The predictability of a feature can be measured by its correlation value with respect to the target class. Correlation value close to 0 means that the feature is not very well correlated to the target class, i.e. it is not an effective feature for prediction. On the other hand correlation value close to -1 or 1 means the feature is strongly correlated to the target class, i.e. it is an effective feature for prediction. Figure 2 shows the box plot of absolute correlation values of each feature with the target class of our dataset, before and after feature selection. The average absolute correlation value of each feature with respect to the target class increased from 0.15(± 0.14) without any feature selection to 0.35(± 0.14) after CfsSubsetEval feature selection. From Figure 2 we can observe that without feature selection the correlation values are positively skewed (meaning more absolute correlation values closer to 0) but after CfsSubsetEval feature selection the correlation values are negatively skewed (meaning more absolute correlation values closer to 1).

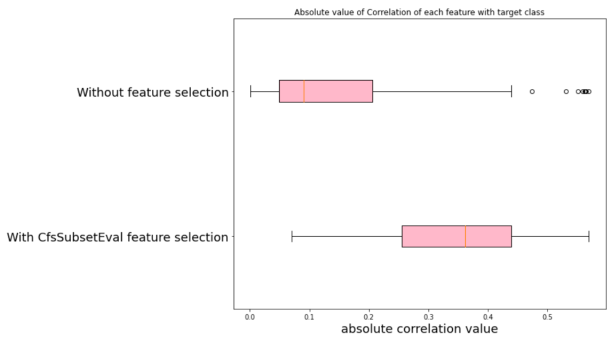

FIGURE 2. Box plot of absolute correlation values of each feature with the target class.

Redundancy among features can be measured using correlation values of each pair of features. A pair of features with correlation value close to 0 has low redundancy whereas correlation value close to 1 or -1 means they have high redundancy with respect to each other. Figure-3 shows the box plot of absolute correlation values of each pair of features before and after feature selection. We found that the average absolute correlation value of each pair of features increased from 0.09(± 0.11) without feature selection to 0.18(± 0.19) after CfsSubsetEval feature selection. However, from Figure 3 we can observe that the desired positive skewness (meaning more absolute correlation values closer to 0) is maintained in both cases.

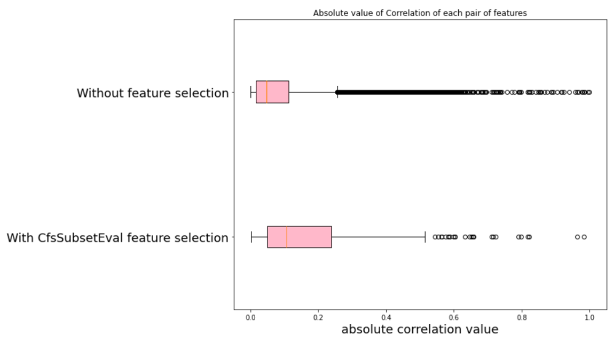

FIGURE 3. Box plot of absolute correlation values of each pair of features.

### D. APPLYING MACHINE LEARNING CLASSIFIERS.

In this study, four machine learning classifiers- Decision Tree, K-Nearest Neighbour, Linear SVM and Random Forest were applied. Hyperparameter tweaking was done for each of the classifiers with stratified 10-fold cross validation on the train set. We used mean value of accuracy, f1-score and AUC, obtained from the 10-fold cross validation, for evaluating a hyperparameter. F1-score was included to address the class imbalance issue and AUC was used for evaluating the appropriateness of a classifier. GridSearchCV [17] was used to exhaustively search over a range of hyperparameter values and find the best one. Finally the classifiers with chosen hyperparameters were evaluated on the test set.

*Decision Tree*

Decision tree based on CART (classification and regression trees algorithm) [18] was used in our study. For measuring impurity of a node in the tree, "gini index" or "entropy" can be used. Max depth, longest length from the root to a leaf node in the tree, was taken as the hyperparameter. Using GridSearchCV [17] we checked for max depth values from 1 to 30 as well as gini and entropy with stratified 10 fold cross validation. Accuracy, f1-score and AUC all three were given as the scoring criteria. Gini index and max depth=17 were chosen by GridSearchCV. Figure 4 shows average accuracy, f1-score and AUC of stratified 10 fold cross validation for different values of max depths, using Gini index.

*K-Nearest Neighbor*

The K-Nearest Neighbour (KNN) classifier is an instance-based classifier with a lazy learning method. It performs predictions on test data based on the classes of 'k' number of nearby training instances. Distance between the test instance

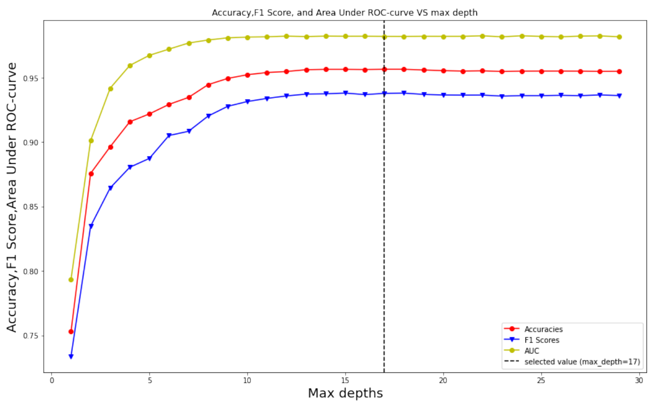

FIGURE 4. Decision tree, stratified 10 fold cross validation average accuracy, f1 score, auc VS max depth (using Gini index).

and instances of the train set can be obtained using various metrics such as- Manhattan distance, Euclidean distance, Minkowski distance etc. For our experimentation we used Minkowski distance with p=2 (which is basically the Euclidean distance) and number of neighbors, 'k' was the hyperparameter. GridSearchCV was used in a similar process as in the decision tree to find optimal k values from 1 to 30 and number of neighbours, k=4 was picked. Figure 5 confirms that number of neighbours, k=4 produces the best combination of average 10 fold cross validation accuracy, f1-score and AUC.

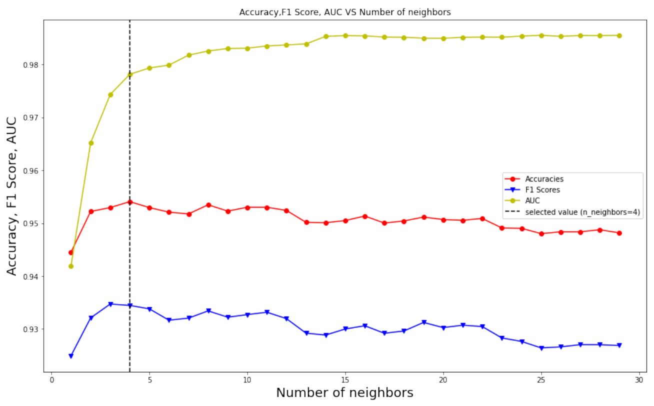

FIGURE 5. KNN, stratified 10 fold cross validation average accuracy, f1 score, auc VS number of neighbors (using Euclidean distance).

*Linear SVM*

Linear SVM classifier works by creating a decision boundary i.e. a straight line separating the instances based on prediction class [19]. The hyperparameter used for tuning the Linear SVM model is the 'C' value. A high C value means fewer margin violations with a smaller margin. Whereas, a low C value means larger margin (meaning the model is more likely to generalize) but higher margin violation. Using GridSearchCV [17] with stratified 10 fold cross validation, we checked for integer values of C from 1 to 30. The scoring criteria was set to accuracy, f1-score and AUC. C=28 was chosen by GridSearchCV. Figure 6 shows that, C=28 produces the best combination of 10 fold cross validation average accuracy, f1-score and AUC.

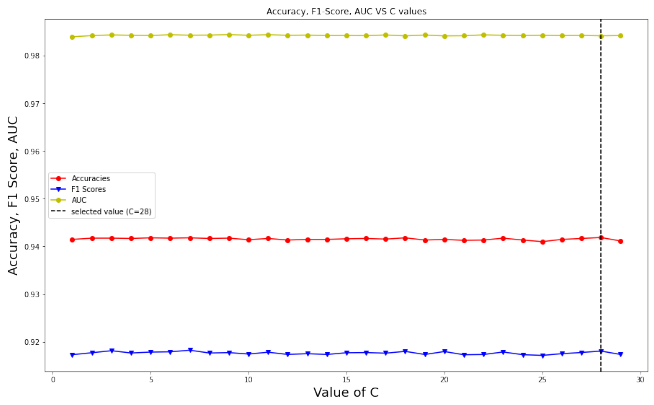

**FIGURE 6.** Linear SVM, stratified 10 fold cross validation average accuracy, f1 score, auc VS values of C.

*Random Forest*

Random Forest is an ensemble of Decision Trees, usually trained via bagging method (or pasting) [19]. The hyper-parameters tweaked for the classifier are n_estimators, the number of trees in the forest and max_features, the number of features to consider for best split. For n_estimators integer values from 1 to 50 were considered. For max_features square root of total number of features, log2 of total number of features and half of total number of features were considered. Using GridSearchCV in the similar process as the classifiers discussed before, the best parameters were obtained. Number of trees, n_estimators = 17 and number of features to consider for best split, max_features = log2 of total number of features were picked. Figure 7 confirms that n_estimator = 17 produces the best combination of average 10 fold cross validation accuracy, f1 score and AUC with max_features set to log2 of total number of features.

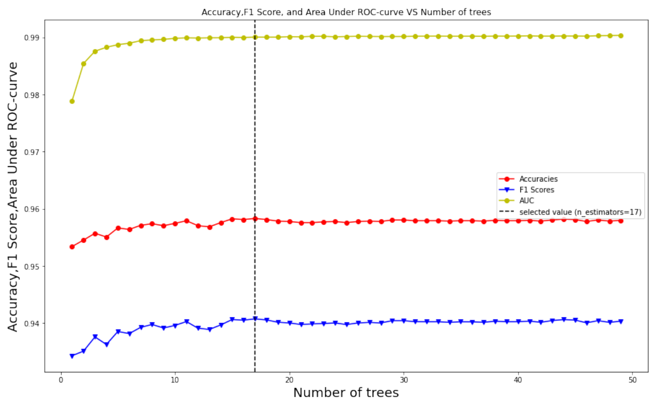

**FIGURE 7.** Random Forest, stratified 10 fold cross validation average accuracy, f1 score, auc VS number of trees (using max_features='log2').

### E. EVALUATION

Finally, each of the four classifiers mentioned were trained with their selected optimal hyperparameter value on the train set and applied to the test set. Note that while hyperparameter tweaking with GridSearchCV, only the train set was used. The test set was completely unseen to the models before evaluation. This was done to depict a more proper picture of how the model would perform in production with real world data.

## IV. RESULTS AND DISCUSSION

In this section we discuss the performance of the four classifiers- Decision Tree, KNN, Linear SVM, Random Forest on both the train and test set. The evaluation metrics- accuracy, precision, recall, f1-score and area under the ROC-curve (AUC) are considered. The formula for these metrics are given in equations(1)-(4). Table 4 shows the performance of each classifier on the train set and Table 5 is for the test set. Figures 8-11 show the train set and test set ROC curve for each of the classifiers.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{1}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{2}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{3}$$

$$F1\_Score = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)} \tag{4}$$

From Table-4, 5 we can see that among the four classifiers, Random Forest had better performance. Apart from the precision value on both train and test set, Random Forest had the highest score for all the other metrics on both train and test set. The accuracy was 0.965 on the train set and 0.963 on the test set. F1 score came out 0.951 on the train set and 0.947 on the test set. AUC was 0.994 on train and 0.993 on test set. The scores on test and train set differ by very little, this indicates that chances of overfitting or underfitting are extremely low for the models. Note that the test set portion of the dataset was completely hidden from the models while training, because in the hyperparameter tuning phase stratified 10 fold cross validation was performed only on the train set. Figures 12-14 show the comparison of performance in train and test set for each classifier.

Table 6 shows a comparison of performance of our best classifier- Random Forest on test set with other notable works. It is to be noted that, the datasets and features used by our work and the comparing works were not identical.

| | Accuracy | Precision | Recall | F1 score | AUC |
|---|---|---|---|---|---|
| Decision Tree | **0.965** | **0.983** | 0.92 | **0.951** | **0.994** |
| KNN | 0.959 | 0.979 | 0.908 | 0.942 | 0.985 |
| Linear SVM | 0.943 | 0.942 | 0.899 | 0.92 | 0.985 |
| **Random Forest** | **0.965** | 0.982 | **0.922** | **0.951** | **0.994** |

TABLE 4. Train set evaluation scores.

| | Accuracy | Precision | Recall | F1 score | AUC |
|---|---|---|---|---|---|
| Decision Tree | 0.961 | 0.961 | 0.93 | 0.945 | 0.985 |
| KNN | 0.958 | **0.965** | 0.916 | 0.94 | 0.982 |
| Linear SVM | 0.946 | 0.937 | 0.912 | 0.924 | 0.984 |
| **Random Forest** | **0.963** | 0.964 | **0.931** | **0.947** | **0.993** |

TABLE 5. Test set evaluation scores.

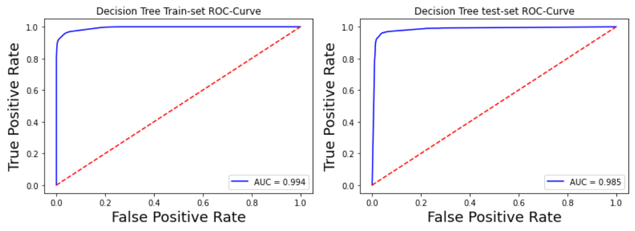

FIGURE 8. Decision Tree train and test set ROC curve.

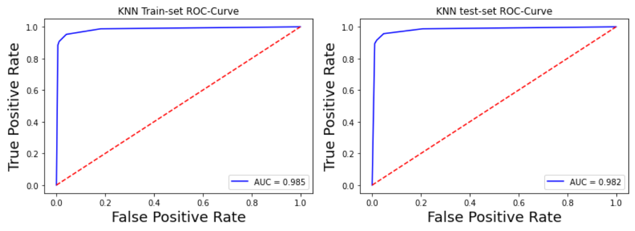

FIGURE 9. KNN train and test set ROC curve.

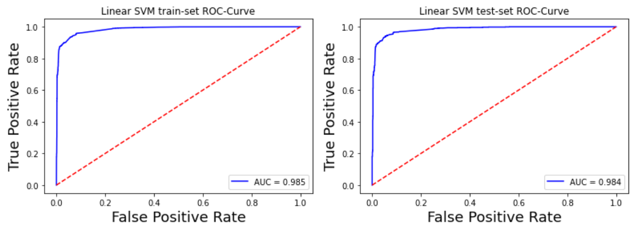

FIGURE 10. Linear SVM train and test set ROC curve.

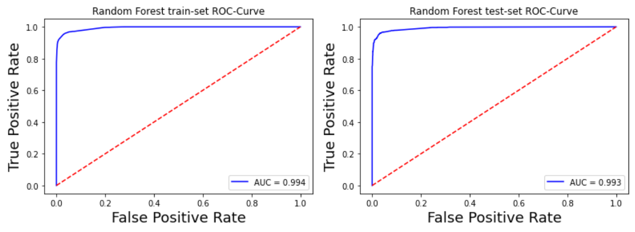

FIGURE 11. Random Forest train and test set ROC curve.

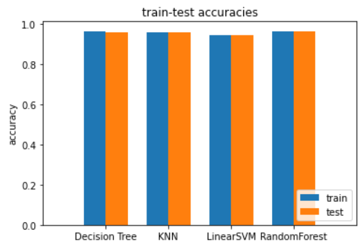

FIGURE 12. Accuracy of classifiers on train and test set.

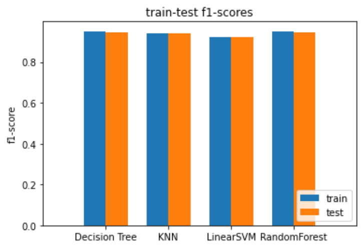

FIGURE 13. F1-scores of classifiers on train and test set.

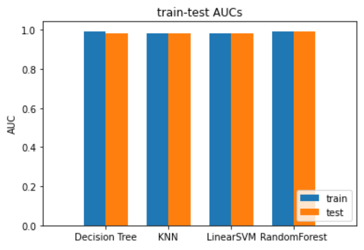

FIGURE 14. AUC of classifiers on train and test set.

| | Accuracy | Precision | Recall | F1 Score | AUC |
|---|---|---|---|---|---|
| [5] | 0.94 | - | - | - | - |
| [20] | 0.85 | - | 0.8 | - | - |
| [9] | - | 0.992 | 0.988 | **0.990** | - |
| [10] | **0.998** | **0.995** | **0.9997** | - | - |
| [8] | 0.987 | 0.979 | 0.995 | 0.987 | - |
| **Proposed Work** | **0.963** | **0.964** | **0.931** | **0.947** | **0.993** |

TABLE 6. Comparison of performance with notable works.

## V. CONCLUSION

Android OS market share is growing at a rapid pace. Not only smartphones, the platform has also integrated with smart TVs, smart watches, smart cars etc. And because of its popularity, ensuring security has become crucial for protecting the privacy of the users. Detecting and preventing the spread of malware applications is a crucial step in ensuring security. In our study we have evaluated machine learning classifiers to detect android malware applications from static features. We have collected two datasets from [4] and systematically conducted experimentation by pre-processing the dataset, feature selection, applying machine learning classifier by hyperparameter tuning with stratified k-fold cross validation on the train set and finally evaluating the models on the test set. We have applied four classifiers- Decision Tree, KNN, Linear SVM and Random Forest and found that Random Forest, an ensemble machine learning method, performs best. Random Forest reported accuracy = 0.963, precision = 0.964, recall = 0.931, f1 score = 0.947 and AUC = 0.993 on the test set.

In future, we would like to test our model on recent Android apps collected from [21]. This would give us insight about how machine learning classifiers trained on existing renowned popular malware repositories perform on new variations of malware families. However, the main challenge for this is to build a static analyzer tool that extracts features from apks and performing the analysis on a bulk of apks which requires a powerful computational machine.

## REFERENCES

[1] Tianyi Gu. Newzoo's global mobile market report: Insights into the world's 3.2 billion smartphone users, the devices they use the mobile games they play. Available at https://newzoo.com/insights/articles/newzoos-global-mobile-market-report-insights-into-the-worlds-3-2-billion-smartphone-users-the-devices-they-use-the-mobile-games-they-play/ (2019/9/17).

[2] Mobile operating system market share worldwide - april 2021. Available at https://gs.statcounter.com/os-market-share/mobile/worldwide (2021).

[3] Raj Samani. Mcafee mobile threat report q1, 2020. Technical report, 2821 Mission College Blvd., Santa Clara, CA 95054, 2020.

[4] Suleiman Y Yerima and Sakir Sezer. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. IEEE transactions on cybernetics, 49(2):453–466, 2018.

[5] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In Ndss, volume 14, pages 23–26, 2014.

[6] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In 2012 IEEE symposium on security and privacy, pages 95–109. IEEE, 2012.

[7] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In International conference on security and privacy in communication systems, pages 86–103. Springer, 2013.

[8] Diyana Tehrany Dehkordy and Abbas Rasoolzadegan. A new machine learning-based method for android malware detection on imbalanced dataset. Multimedia Tools and Applications, pages 1–22, 2021.

[9] Zhuo Ma, Haoran Ge, Yang Liu, Meng Zhao, and Jianfeng Ma. A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE access, 7:21235–21245, 2019.

[10] Prerna Agrawal and Bhushan Trivedi. Machine learning classifiers for android malware detection. In Data Management, Analytics and Innovation, pages 311–322. Springer, 2021.

[11] Fei Tong and Zheng Yan. A hybrid approach of mobile malware detection in android. Journal of Parallel and Distributed computing, 103:22–31, 2017.

[12] Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. Semantics-based online malware detection: Towards efficient real-time protection against malware. IEEE transactions on information forensics and security, 11(2):289–302, 2015.

[13] Shaikh Bushra Almin and Madhumita Chatterjee. A novel approach to detect android malware. Procedia Computer Science, 45:407–417, 2015.

[14] Fabio Martinelli, Francesco Mercaldo, and Andrea Saracino. Bridemaid: An hybrid tool for accurate detection of android malware. In Proceedings of the 2017 ACM on Asia conference on computer and communications security, pages 899–901, 2017.

[15] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. ACM SIGKDD explorations newsletter, 11(1):10–18, 2009.

[16] Mark Andrew Hall. Correlation-based feature selection for machine learning. 1999.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. the Journal of machine Learning research, 12:2825–2830, 2011.

[18] Dan Steinberg. Cart: classification and regression trees. In The top ten algorithms in data mining, pages 193–216. Chapman and Hall/CRC, 2009.

[19] Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019.

[20] Mohsen Kakavand, Mohammad Dabbagh, and Ali Dehghantanha. Application of machine learning algorithms for android malware detection. In Proceedings of the 2018 International Conference on Computational Intelligence and Intelligent Systems, pages 32–36, 2018.

[21] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.