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. So only developing of ML model which can detect malware is not efficient. It is important is check the working of the app. For this an android app is build using android studio which can be installed in android phones and can detect malware efficiently.

Keywords— Android Malware detection, Machine Learning, Static Analysis of Android Malware, Android Studio

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

1.1 Challenges

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 percent 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.

According to the IDC report shown in the 2016-year, Android market share will reach 85.3 percent until the end of 2016. Android has been more and more indispensable with its open-source character and advantages of free in our daily life. However, the number of malicious software is also growing rapidly. Therefore, how to detect the Android malware with the high accurate rate is a hot issue. Traditional detection approach based on signature is widely used both in Android devices and PC platform by extracting the signature from APK and comparing with the malicious signature in the virus database, however, this approach is limited to detect unknown malwares which are not existed in the virus database. In order to address this question, the previous researchers find that there are two techniques for the unknown Android malware analysis: static analysis and dynamic analysis. Static analysis, occurs before the Android application is installed, is a technique based on checking the contents of the APK by means of reverse engineering. Different from the static analysis, dynamic analysis monitors the running state of the Android application in the virtual environment. With the development of machine learning technology, machine learning approaches are also mentioned to classify obtained observations as either benign or malicious by collecting different signal features and events from the applications and the system, however the raw features may exist irrelevant or redundant features that may lead to a wrong result, so feature selection process is vital. Feature extraction and feature selection are the crucial steps which determine the accuracy rate of the classifier. After that it will be easier to achieve the detection result with the classifier.

1.2 Motivation

Due to the increasing openness and popularity, android phones have been an attraction to most of the malicious applications and an attacker can easily embed its own code into the code of a benign application. Therefore, malwares attacking the android application are growing at an alarming rate and under these circumstances, security of the devices and the assets these devices allow access to, be at stake. In addition, android itself has some distinct characteristics and limitations due to its mobile nature. Therefor it is important is detect malware in any android system. The efficient way of detecting malware is using machine learning and also only ml model generation is not enough there also need an android app which can effectively find malicious apps in android phone.

LITERATURE REVIEW

2.1 Related Work

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.

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 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 percent of malwares with 1 percent 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 percent and 97.8 percent recall was reported.

In [8] class imbalance issue of malware datasets was addressed. Synthetic minority oversampling(SMOTE), random under sampling 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 percent, f1-score 98.69 percent and MCC score 97.39 percent.

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 percent, precision 99.15 percent, recall 98.82 percent and f1-score 98.98 percent. 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 percent accuracy, 99.52 percent precision and 99.54 percent recall.

In dynamic analysis, 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. 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 percent along with FPR less than 4 percent if parameters are chosen correctly. However, this method cannot be used for real time malware detection because of big data processing and privacy protection on outsourcing data. In 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 percent, AUC of 0.997 and FPR of 1.2 percent. In 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 the authors propose a malware detection system using both static and dynamic analysis. Its malware detection accuracy reached up to 99.7 percent. But it requires high consumption of local resources.

2.2 Problem Statement

Nowadays most of the android apps are subjected to malicious activities without our knowledge. And it is important to remove those malicious apps from our android phone otherwise such apps can spy our regular activities and leak our personal information which leads to a high risk. So it is important to detect those malicious apps in android phone and remove it as soon as possible. In this paper, those malicious apps are detected by using the permissions taken by the app with or without user knowledge using different machine learning models. From those different ML models best one is chosen and used in the backend of the android app.

2.3 Research Objectives

The purpose of malware analysis is to obtain and provide the information needed to rectify a network or system intrusion. The main objective of the paper is to detect the malicious apps present in any android phone by using machine learning algorithms with the help of apk's, permission list, intents of all apps present in that android phone.

METHODOLOGY AND FRAMEWORK

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, hyperparameter 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.

3.1 Obtaining Dataset

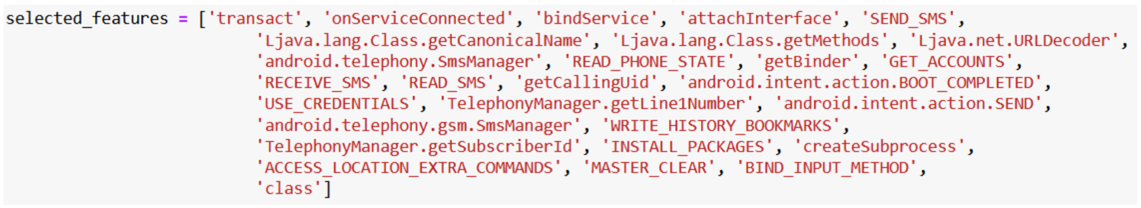
Two dataset was used namely Drebin and Malgenome. Both the datasets contain 215 features, consisting of four feature sets: (i)API calls- method names extracted from apk's .dex file, (ii)Manifest permissions- permissions declared in apk's AndroidManifest.xml file, (iii)Intents- used for communication with other apps/services declared in the apk's Android- Manifest.xml and (iv)Command found in the apk's .dex, .jar, .so, .exe files shows that these set of features work well for machine learning. Features were extracted by 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

3.2 Preprocessing Dataset

As mentioned, before we collected two datasets- Drebin- 215 and Malgenome-215 from 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. 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.

3.3 Feature Selection

Feature selection on the train set was done using correlation-based feature selection (CfsSubsetEval) in Weka. 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. Out of the 208 features of the train set, 27 were selected. 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.



```
selected_features = ['transact', 'onServiceConnected', 'bindService', 'attachInterface', 'SEND_SMS',
                     'Ljava.lang.Class.getCanonicalName', 'Ljava.lang.Class.getMethods', 'Ljava.net.URLDecoder',
                     'android.telephony.SmsManager', 'READ_PHONE_STATE', 'getBinder', 'GET_ACCOUNTS',
                     'RECEIVE_SMS', 'READ_SMS', 'getCallingUid', 'android.intent.action.BOOT_COMPLETED',
                     'USE_CREDENTIALS', 'TelephonyManager.getLine1Number', 'android.intent.action.SEND',
                     'android.telephony.gsm.SmsManager', 'WRITE_HISTORY_BOOKMARKS',
                     'TelephonyManager.getSubscriberId', 'INSTALL_PACKAGES', 'createSubprocess',
                     'ACCESS_LOCATION_EXTRA_COMMANDS', 'MASTER_CLEAR', 'BIND_INPUT_METHOD',
                     'class']
```

Figure 3.3.1: Selected Features

Figure 3.3.2 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 3.3.2 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).

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. Below figure 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($\pm$ 0.11) without feature selection to 0.18($\pm$ 0.19) after CfsSubsetEval feature selection. However, from below figure we can observe that the desired positive skewness (meaning more absolute correlation values closer to 0) is maintained in both cases.

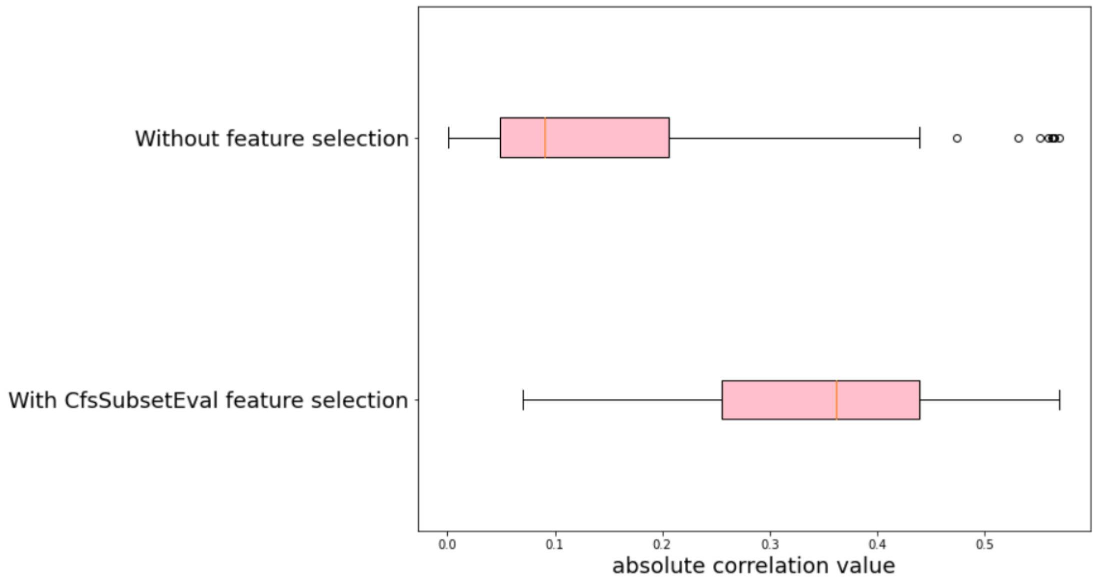


Figure 3.3.2: Box plot of correlation of target class and selected features vs correlation of target class and all features

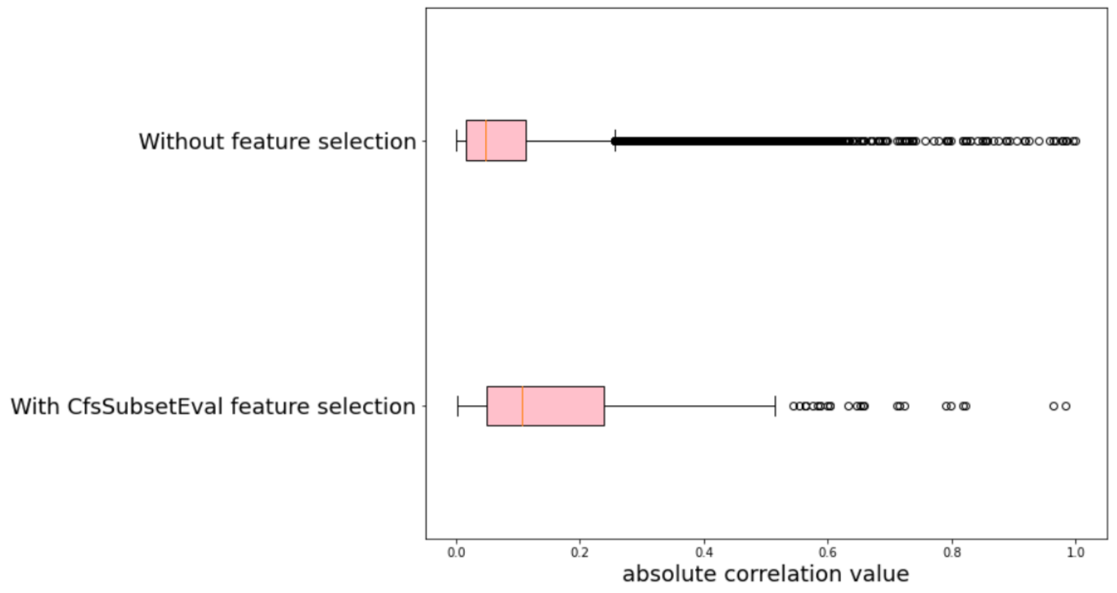


Figure 3.3.3: Box plot of correlation between each pair of selected features vs correlation between each pair of all features

3.4 Machine Learning Classifiers

In this study, four machine learning classifiers- Decision Tree, K-Nearest Neighbor, 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.

3.4.1 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. C=28 was chosen by GridSearchCV. Below figure shows that, C=28 produces the best combination of 10-fold cross validation average accuracy, f1-score.

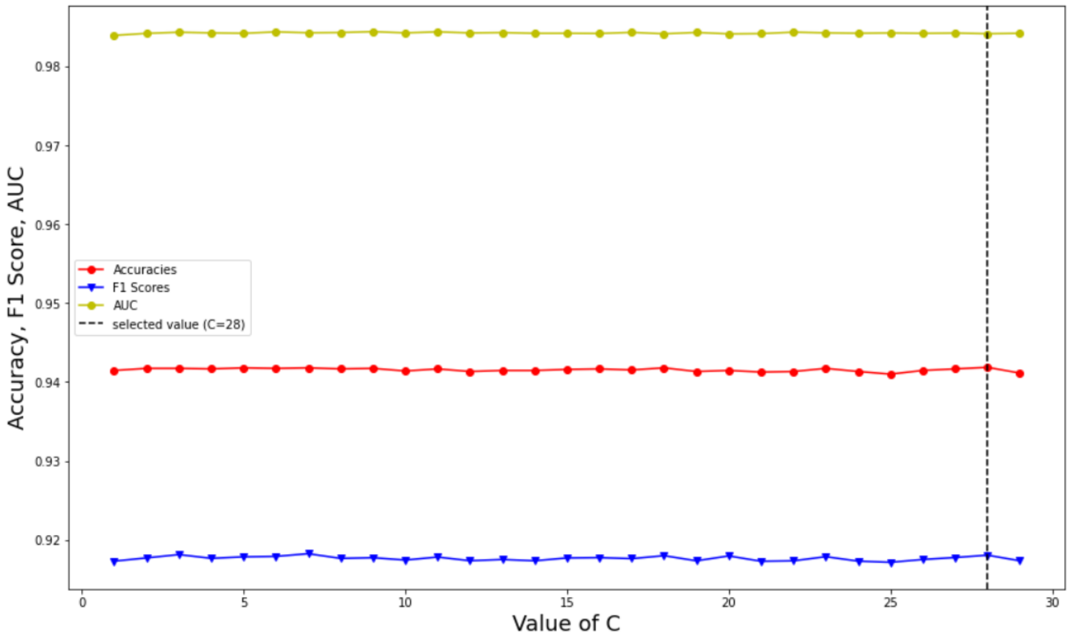


Figure 3.4.1: Linear SVM

3.4.2 KNN

The K-Nearest Neighbour (KNN) classifier is an instancebased 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 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=2 was picked. Below figure confirms that number of neighbours, k=2 produces the best combination of average 10-fold cross validation accuracy, f1- score.

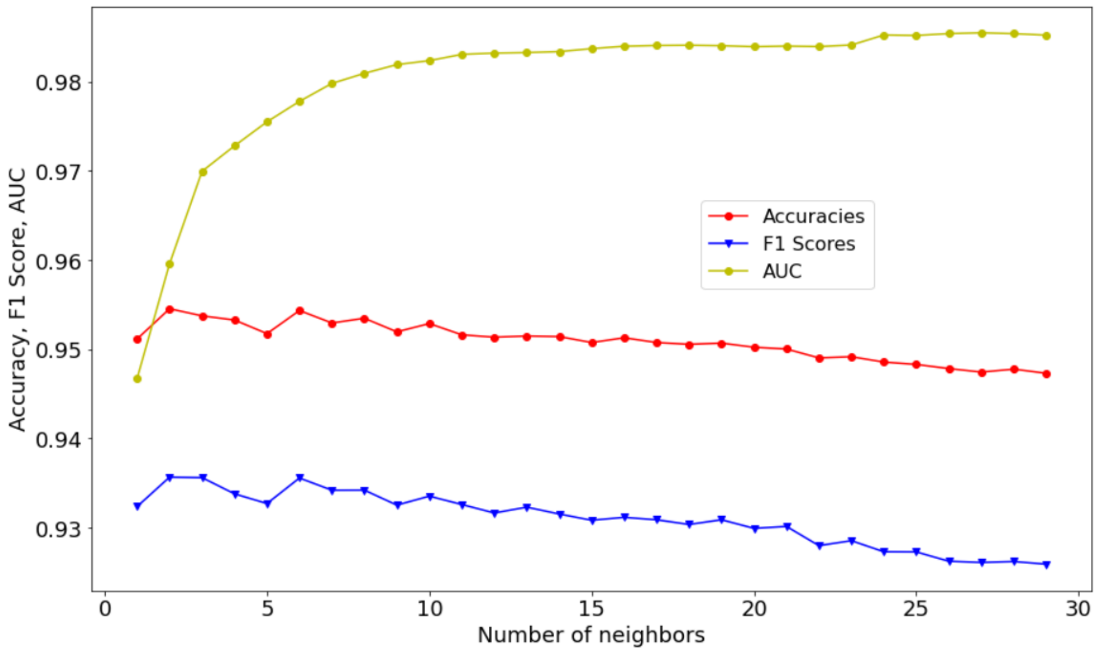


Figure 3.4.2: KNN

3.4.3 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. Below figure shows average accuracy, f1-score and AUC of stratified 10-fold cross validation for different values of max depths, using Gini index.

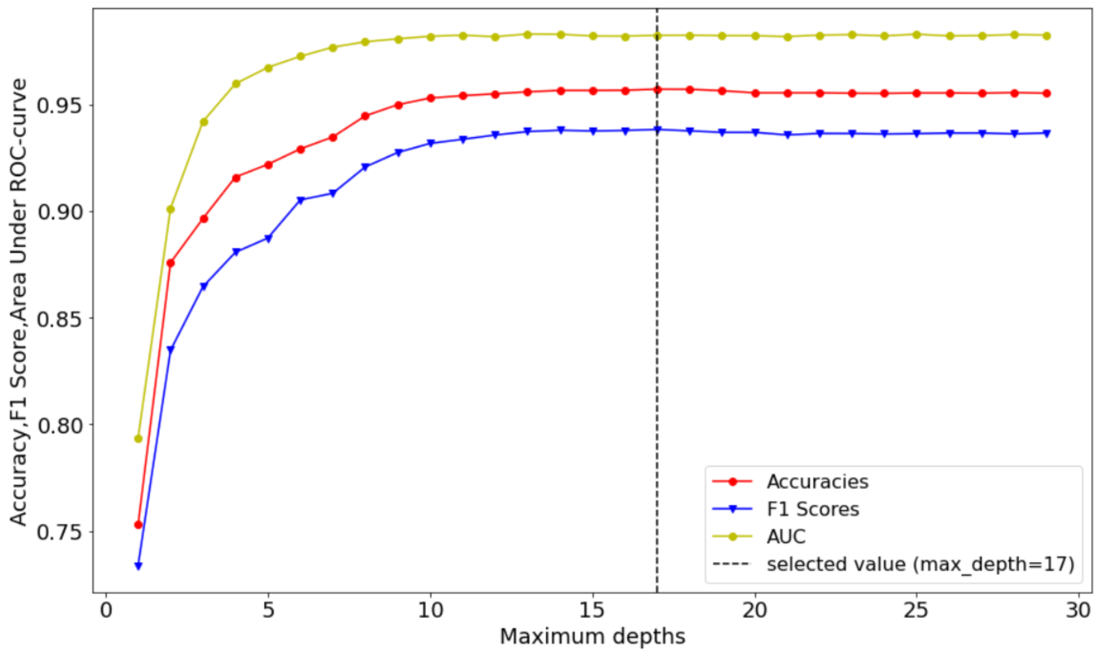


Figure 3.4.3: Decision Tree

3.4.4 Random Forest

Random Forest is an ensemble of Decision Trees, usually trained via bagging method (or pasting) [19]. The hyperparameters 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. Below figure 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.

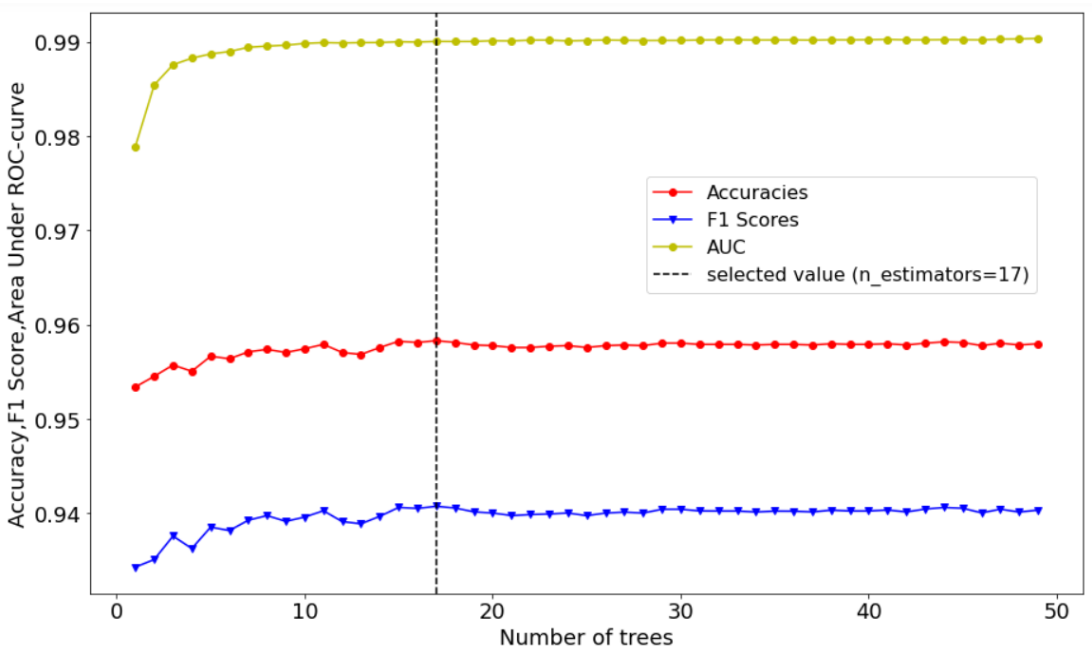


Figure 3.4.4: Random Forest

3.5 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.

3.6 Creating an android app

The android app is created by using android studio. Android Studio provides app builders with an integrated development environment (IDE) optimized for Android apps. The official language for Android development is Java. Large parts of Android are written in Java and its APIs are designed to be called primarily from Java. So Java is used as a backend for the development of the android app.

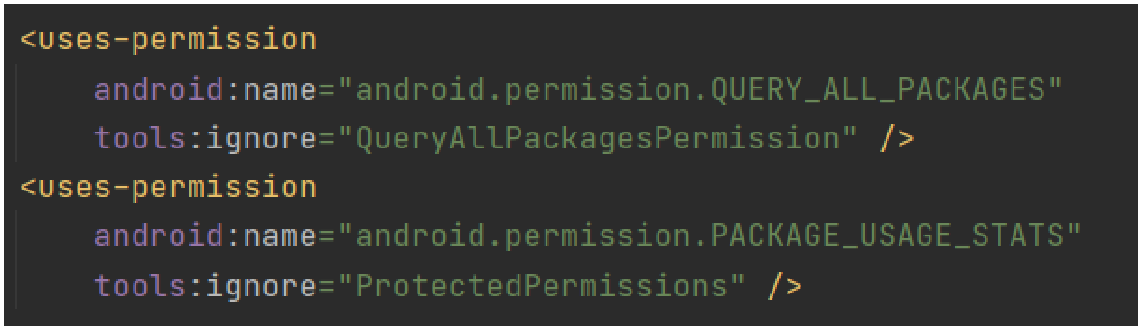
3.6.1 Frontend development

The front-end part helps to design the app that is how the app need to be look like. This section consist 2 xml files : one is the main layout of the app and another helps in representing the list view of the layout.

In main layout is created with the name activity main.xml file format and Linear Layout is used to represent the layout view. LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. Inside the Linear Layout, 6 TextView and 1 ListView are declared to represent all the information of the results of the app. The first TextView, helps to tell whether the android phone is safe or not that is the final result of the app. The second TextView helps to declare or view the “Result” keyword in the screen of the app. The third TextView helps to tell the total number of apps detected in the particular android phone. The fourth TextView helps to tell the total number of apps that are detected as malicious. The fifth TextView helps to tell the total number of all that are detected as non-malicious. The sixth TextView helps to view the “App List” keyword on the app screen. The ListView helps to list all the app names present in the android phone with a subtitle which indicates whether the app has any thread or not. The list view xml file has declared with 2 TextView

3.6.2 Android Manifest

During the development of the android app, there need some changes in the android manifest file of the created android app using android studio. Those are need to give access by the user for accessing all the package details of all the apps present in the android phone.



```
<uses-permission
    android:name="android.permission.QUERY_ALL_PACKAGES"
    tools:ignore="QueryAllPackagesPermission" />
<uses-permission
    android:name="android.permission.PACKAGE_USAGE_STATS"
    tools:ignore="ProtectedPermissions" />
```

Figure 3.6.1: Android Manifest file

Also need to get the access package usage stats which helps to retrieve information from the each app present in the android phone.

3.6.3 Backend Development

The backend of the android app is developed in java. Two java files are created which helps in the working of the android app. MainActivity java file is the main program file where the operations take place. First all the packages of the android phone is obtained through getPackageManager() function. Here package refers to the app name and its details. Then next step is obtaining list of all permissions taken by all android apps present in the android phone. For this task, getPackageInfo() fuction is used which helps in determining list of all permissions of the android app. Then the load the best resultant ML model from the developed ML classifiers. In our case, RandomForest classifier is used to determine whether the corresponding android app is malicious or not. From the developed RandomForest model, feature importance value is taken and implemented here. There are 27 features, so there are 27 weights of each feature. Sum of all the weights corresponds to 1.

The below figure represents how the malware rate of each feature is calculated. If the malware rate is near to 0 then the app is subjected to be non-malicious else the app may have some thread. So such apps need to be removed immediately from the android phone.

The second Java file helps to create the list view of all apps by creating an ListAdapter. ListAdapter helps to structres the list view of the app.

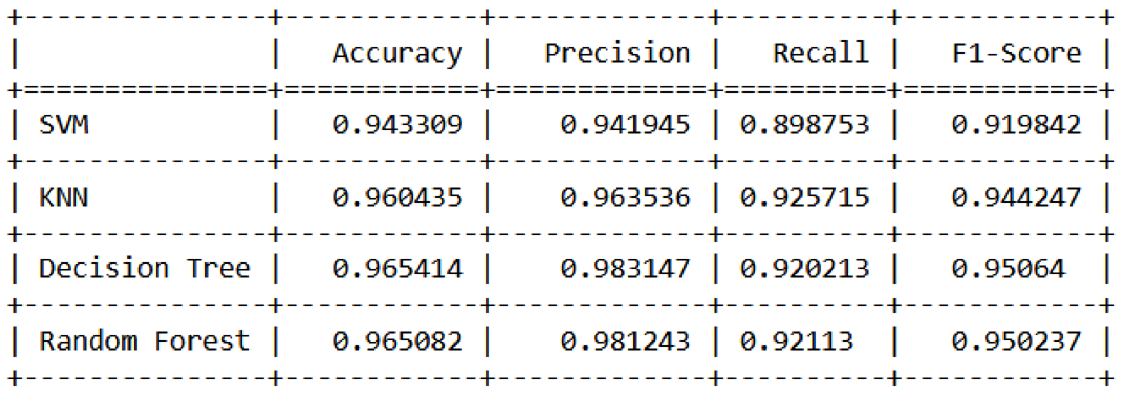
3.6.4 App Icon

The app icon is created by using other online software and saved it as a logo.jpeg file. This jpeg file is imported inside drawable folder. Then in the android manifest file, change the value of icon inside the application to “@drawable/logo”, so that the corresponding image would appear as app icon.

RESULTS AND ANALYSIS

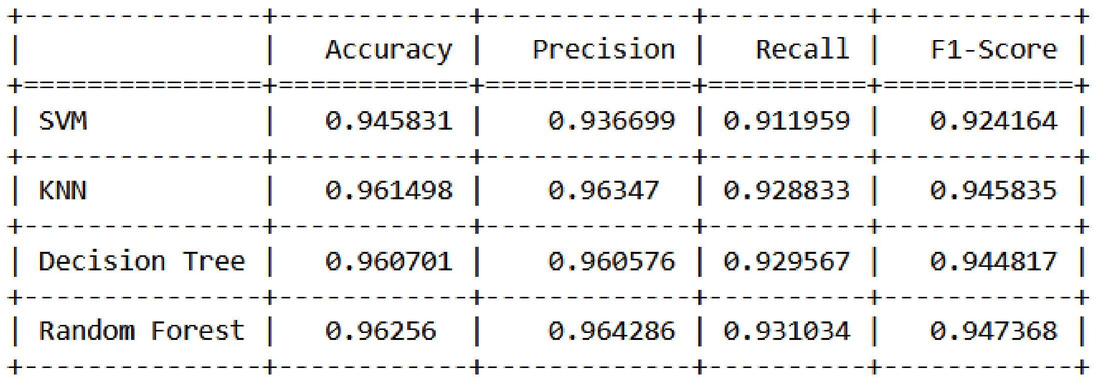
4.1 Results of ML model

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 metricsaccuracy, precision, recall, f1-score and area under the ROCcurve (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 |   Precision |   Recall |   F1-Score |
+===============+============+=============+==========+============+
| SVM           |   0.943309 |    0.941945 | 0.898753 |   0.919842 |
+---------------+------------+-------------+----------+------------+
| KNN           |   0.960435 |    0.963536 | 0.925715 |   0.944247 |
+---------------+------------+-------------+----------+------------+
| Decision Tree |   0.965414 |    0.983147 | 0.920213 |   0.95064  |
+---------------+------------+-------------+----------+------------+
| Random Forest |   0.965082 |    0.981243 | 0.92113  |   0.950237 |
+---------------+------------+-------------+----------+------------+
```

Figure 4.1.1: Results of Train set



```
+---------------+------------+-------------+----------+------------+
|               |   Accuracy |   Precision |   Recall |   F1-Score |
+===============+============+=============+==========+============+
| SVM           |   0.945831 |    0.936699 | 0.911959 |   0.924164 |
+---------------+------------+-------------+----------+------------+
| KNN           |   0.961498 |    0.96347  | 0.928833 |   0.945835 |
+---------------+------------+-------------+----------+------------+
| Decision Tree |   0.960701 |    0.960576 | 0.929567 |   0.944817 |
+---------------+------------+-------------+----------+------------+
| Random Forest |   0.96256  |    0.964286 | 0.931034 |   0.947368 |
+---------------+------------+-------------+----------+------------+
```

Figure 4.1.2: Results of Test set

4.2 Results of Android App

4.2.1 App Icon

The below figure represents the app icon of the developed android app. And the name of the android app is kept as “MalDect”. This icon is created by using online image processing websites.

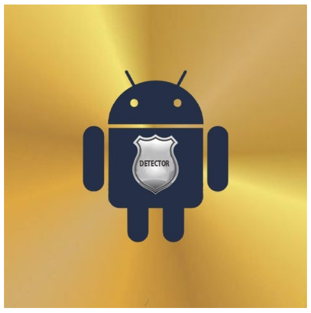
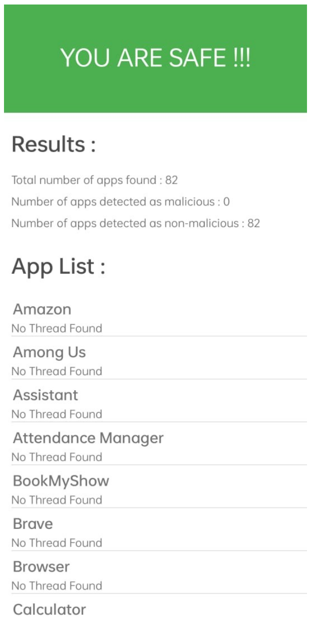


Figure 4.2.1: App Icon

4.2.2 App Layout

The below figure represents layout of the app that is screenshot of the app running.



YOU ARE SAFE !!!

Results :

Total number of apps found : 82
Number of apps detected as malicious : 0
Number of apps detected as non-malicious : 82

App List :

Amazon
No Thread Found
Among Us
No Thread Found
Assistant
No Thread Found
Attendance Manager
No Thread Found
BookMyShow
No Thread Found
Brave
No Thread Found
Browser
No Thread Found
Calculator

Figure 4.2.2: App Layout

The first part represents your android phone is safe or not. If your android phone is not safe then the first part projects “YOU ARE NOT SAFE” with red background. The second part represents the number of total apps, number of apps suspected as malicious and number of apps suspected as non-malicious. The third part represents list of all apps present in the android phone with it is having thread or not.

CONCLUSIONS AND FUTURE WORK

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 on the test set and we also successfully build the android app using java in android studio.

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. Also we will build the android app in more efficient way such that it can automatically detect all apps without opening MalDect and also gives report of every newly installed app in the android phone before opening the newly installed app.

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.

[2] Mobile operating system market share worldwide - april 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 apilevel 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. Acombination 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 HWitten. 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.