<https://www.researchgate.net/profile/Suman_Tiwari4/publication/330133778_An_Android_Malware_Detection_Technique_Based_on_Optimized_Permissions_and_API/links/5c304c79458515a4c70d3767/An-Android-Malware-Detection-Technique-Based-on-Optimized-Permissions-and-API.pdf>

**Android Malware Detection using Static and Dynamic Features**

Kaustubh

Computer and Information Science

University of Massachusetts , Dartmouth

North Dartmouth, MA, USA

kaustubh@umassd.edu

**Dataset:** [**https://drive.google.com/file/d/1Dcuogqdsqi5I7vDXOvPiogSxe3JNCnFf/view**](https://drive.google.com/file/d/1Dcuogqdsqi5I7vDXOvPiogSxe3JNCnFf/view)

**Dataset:** [**Android Malware Dataset (CIC-AndMal2017)**](https://www.unb.ca/cic/datasets/andmal2017.html)

***Abstract:* As the quantity of cell phone clients increases in terms of billions consistently, and clients currently store individual and sensitive data on their cell phone which gives a gigantic stage for hackers to take clients critical and sensitive data. We have proposed a strategy to distinguish android malware utilizing permissions and API that used in an android application. We have created two kinds of feature vectors named as static and dynamic vectors. Through evaluation of the various algorithms, we have got 99.5 % accuracy in DecisionTree Classifier.**

***Keywords***

Android Malware Detection, Machine Learning, Static, Dynamic, Features

1. INTRODUCTION

ANDROID frameworks have increased expanding ubiquity in advanced cells and other versatile insightful terminals in ongoing years. Terribly, the aggregation advancement and open nature of the stage have likewise pulled in a tremendous number of malware engineers. As indicated by a secure list [1], Kaspersky identified 230 million remarkable vindictive and possibly undesirable articles, alongside 870 thousand pernicious establishment bundles in the second from the last quarter of 2019. An agent arrangement is Gustuff [2], which phished qualifications and mechanize bank exchanges for more than 100 financial applications and 32 digital currency applications. The staggering impact is troublesome to get fast and viable control, as the manual analysis is time-devouring and it puts high requests on the experience of security experts.

To control the expanding spread of Android malware, scientists have proposed a few answers for programmed identification. The majority of the current malware discovery strategies basically make a judgment about whether an application is malevolent or then again not, and a portion of the techniques connect delicate authorizations furthermore, other data to the end-product. For security experts, it needs direct proof to help the judgment. To settle this issue, some location strategies hope to trigger as numerous conditions as conceivable by methods for UI (User Interface) associations or programmed testing instruments like Monkey [5]. Such techniques can get irregular practices, yet they still require exertion to discover noxious code for itemized analysis. What's more regrettable is that a few stunts can without much of a stretch detour these techniques, such as the purposely postponed dispatch of malignant projects, the malevolent practices set off by some particular organization information bundles or in some cases even a straightforward login interface

In light of Trend Micro™ Mobile App Reputation information, 80% of the 1 million applications are malevolent in nature [13]. Google has Released Google Bouncer, which is utilized to filter all the use of the Android Play store consequently, however, Scientists guaranteed that it is conceivable to the sidestep Bouncer security component likewise [13]. Thus, any android telephone over the web is on high dangers and have the danger to their protection and security. Static and Dynamic [4] Analysis are three fundamental techniques utilized for malware discovery for Android portable gadgets. The static investigation is the most mainstream strategy for malware location and better for primer level analysis

since it investigates android application before it is introduced in the client gadget and by and large values for the beginning level screening of android application, Where on account of dynamic analysis application should be introduced in the framework.

The dynamic analysis need to run consistently inside client gadgets to check the runtime conduct of the android application which devours more handling limit and force than the static analysis approach, consequently, it is smarter to give better security at the section level, which stops the establishment of malware into client gadget, because of these reasons, we have supported and zeroed in our exploration deal with static analysis strategy instead of dynamic investigation method.

1. LITERATURE SURVEY

After a writing review, we saw that just about 80 percent of paper depended on the static investigation. There is a ton of work done around there as canvassed in beneath area, and still should be investigated more to acquire higher exactness with least number of features, in light of the fact that less number of features lessen preparing and testing season of relapse and grouping and gives a quicker reaction.

Lichao Sun et. all proposed SigPID[1] in which they have broken down consents separated from Androidmanifest.xml. They have utilized three degrees of information pruning(MLDP - staggered information pruning) techniques to channel out consents, which encourages them to eliminate extra consents without influencing exactness. The principal level of pruning is with PRNR(Permission Ranking with Negative Rate) which is utilized to discover just those consents which offer more for malware recognition. The second degree of pruning is SPR(support based consent Ranking) if uphold for any consent is excessively low, at that point it is erased from malware discovery features. Third level pruning is PMAR (Authorization mining with Association rule) which eliminates features on the off chance that they generally meet up ex. READ\_SMS and WRITE\_SMS. After all degrees of pruning, they have acquired just 22 authorizations that are close to the equivalent precision given by most perilous 24 consent gave by Google.

Xiang Li et. al proposed [3], have proposed a static analysis approach dependent on consents from show document, also, broke down 37 most huge authorizations utilizing ChiSquare include choice strategy, and characterized utilizing Naive Bayes classifier.

Ansul Arora et al.[5] has proposed malware identification, which recognizes malware dependent on network traffic investigation.

Traffic features are separated and positioned with the utilization of

Data increase and chi-Square test. They streamlined the

number of features under 9 features which lessen preparing season of the classifier and Naive Bayes classifier is utilized for grouping.

Kun Wang et. al [6] proposed Mmda dependent on metadata in show record. They have extricated authorizations, all out number of authorizations utilized by one example, equipment highlight set, Receiver activity set and number of activities pronounced by one example. They have removed complete 122 features and ordered against different classifiers among which Random Forest gives them 94% exactness.

Alatwi et al.[11] has proposed malware recognition based on the class of utilization. They said the considerate application under the same class utilizes a typical arrangement of features where malignant applications utilize phenomenal features which are extraordinary to their class. They have utilized 2775 features, what's more, ordered utilizing Support vector machines. Sayfullina et al. [9] Proposed a strategy that employs Hashes, Manifest strings, consents, DEX string (APIs) highlight vector. They have extricated 9.9 million features decreased it to 10,000 features utilizing dimensionality decrease, also, arranged utilizing support vector machine which gives them 99.91% exactness.

Cloud-based analysis ScanMe[8] is proposed by Hanlin Zhang which provides new guidance for Android malware investigation. They have proposed a cloud-based malware identification procedure where they proposed to send APK over the cloud and create a report about given APK record is sent

to the cell phone.

Sujithra et al[19] has proposed android malware identification utilizing upgraded arbitrary backwoods classifiers and PSO what's more, GA(Genetic Algorithm). They have separated consent set from show document and utilized IG(information gain) to get just those features which are more gainful for malware identification. They have utilized K-implies bunching calculation to eliminate features which don't have a place with any group . These features are then applied to Random Forest, Random timberland with GA and Random Forest with PSO where Random Forest with PSO beats than another and gives 88.4% precision.

Following Table - 1 gives an outline of not many Literature

Overview done around there up until now.

|  |  |  |  |
| --- | --- | --- | --- |
| **Title/**  **Author** | **Description** | **Analysis** | **Accuracy** |
| SIGPID[1 ] | Multilevel data pruning to optimize permissions | Does not work on skewed dataset Optimized to 22 permissions | 91.34% |
| Xiang Li[3] | -Permissions are extracted and ranked using chiSquare test. | -Optimized to 37 permissions | 82.48% |
| Mmda [6] | -Permission sets, no of permissions, Hardware features, Receiver action, No of Receiver actions | Optimized to 122 features | 94% |
| Sayfullina [9] | PERM, STR,DEX,HASH | Too large feature vector | 99.91% |
| Sujithra[1 9] | Permissions used as features. | IG,KNN,PSO are used for optimization. | 88.4% Using PSO |
| Arora[5] | Network traffics are captured using a sniffer | Find the optimum number of features. - Can not detect encrypted traffic, obfuscated malware | 87.25% using 22 features |
| ScanMe [8] | cloud based sandbox perform Static and dynamic analysis | Processing and power capacity of device is not used. -Network bottleneck | 86% |

1. METHODOLOGIES

We are using static features and dynamic features of an APK and applying various machine learning algorithm to check which algorithm is performing better on our extracted feature dataset. However, In the beginning, we do not have such feature matrix data but instead of it, we only have a collection of APKs, labeled as benign and malign. Hence, we broke down the whole pipeline as below.

1. Decompiling APKs
2. Extracting static features from APKs
3. Extracting dynamic features from API
4. Preparing feature matrix
5. Training on different algorithms
6. Evaluation

After following the mentioned pipeline above, we will have a classifier that can predict the APK’s authenticity based on its static and dynamic features.

1. Decompiling APKS

To decompile our APKs. we are using androguard library which are intent to extract useful features like permissions, operation codes, static credentials, strings etc with use of the APKTool java library.

APKTool is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc.

It is NOT intended for piracy and other non-legal uses. It could be used for localizing, adding some features or support for custom platforms, analyzing applications and much more.

1. Extracting static features from APKs

Static features in APK means those observations that are already available in the application. No need to do extra feature engineering to evaluate them. Some of those features are Permissions, Operation Codes, Static Strings, Package Name etc. We are counting on the static features because, if you observe the multiple malware APKs, you will get to know that the hacker's main intention is to hack into the android system and not into android development so they don’t follow the basic android development best practices guidelines. Hence they create the package name like mse.black.1233 etc instead of the standard one like com.companyname.producname.

Similarly, permission used in app also plays a key role as like the hacker mainly wants to get access of your storage part and system so READ\_EXTERNAL\_STORAGE, INTERNET, READ\_CONTACTS, READ\_SMS are some of critical permissions that being used in malware APKs..

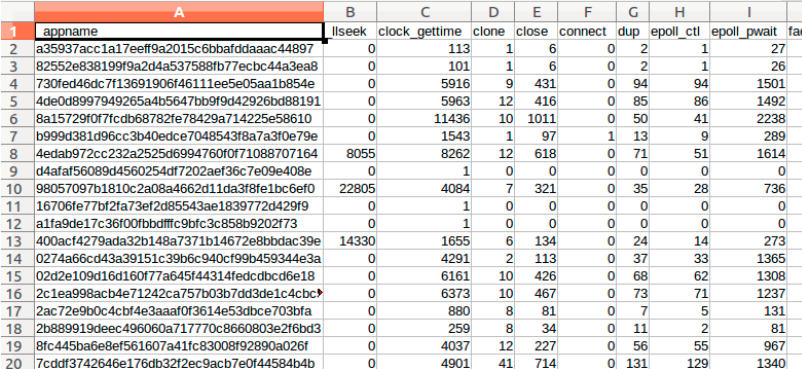


Fig 1. Tabular dataset view of extracted features.

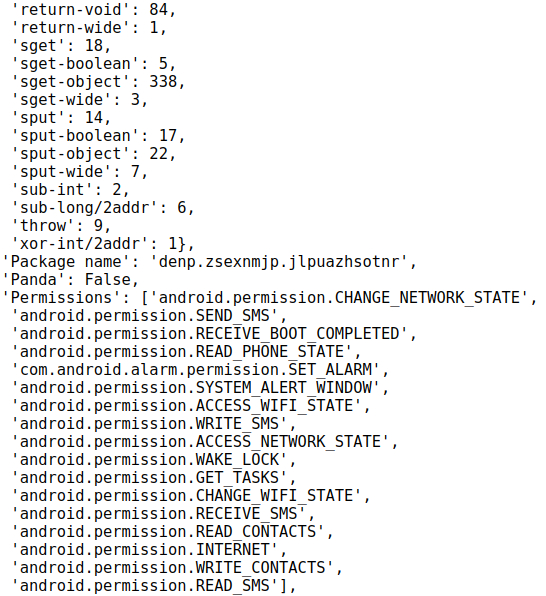


Fig 2. snippet of some features extracted from APKs.

Running the python script 'prepare\_dataset.py', We have decompiled all the APKs available in the dataset and get as many features that we can grab and are relevant to detect malware or phishing Apps.

In android app development, The main part of getting into someone's privacy is allowing the dangerous permissions that application ask to the user. Such as permission like Contacts, Storage, Get Running Tasks etc. By using these permissions various apk grabs the confidential data from the app and may misuse it or sell it.

So We have a full list of permissions available in android os so loading them and creating a matrix for permission in a way that we can feed it into the classifier.

1. Extracting dynamic features from API

To extract the dynamic features of the APK, we have used virus toal’s API services to extract the dynamic features from APK. To do so we have uploaded all the APKs recursively on cloud and got its features in response.

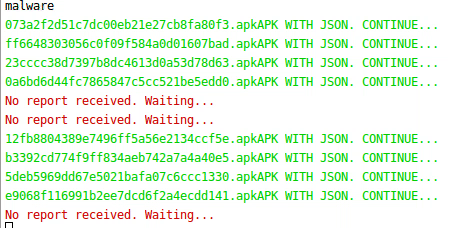


Fig 3. Sending APKs for dynamic analysis

1. Preparing feature matrix

After doing the static and dynamic feature extraction of an application. Now the main task is to prepare the feature matrix of app by combining static features matrix and dynamic feature matrix into one matrix.

1. Training on different algorithms

Finding a good or best mapping function for a dataset is a type of search problem. We test different algorithms and test algorithm configurations that define the shape of the search space and give us a starting point in the search space. We then run the algorithms, which then navigate the search space to a single model.

Adding randomness can help avoid the good solutions and help find the really good and great solutions in the search space. They allow the model to escape local optima or deceptive local optima where the learning algorithm might get such, and help find better solutions, even a global optima.

Hence, to check on which algorithm works best on our feature matrix, we will apply the top most used classical machine learning algorithms which are as follows.

**SVM (Support Vector Machine)**

It is an order technique. In this calculation, we plot every information thing as a point in n-dimensional space (where n is the quantity of highlights you have) with the estimation of each component being the estimation of a specific facilit.

For instance, in the event that we just had two highlights like Height and Hair length of an individual, we'd initially plot these two factors in two-dimensional space where each point has two facilitates (these coordinates are known as Support Vectors)

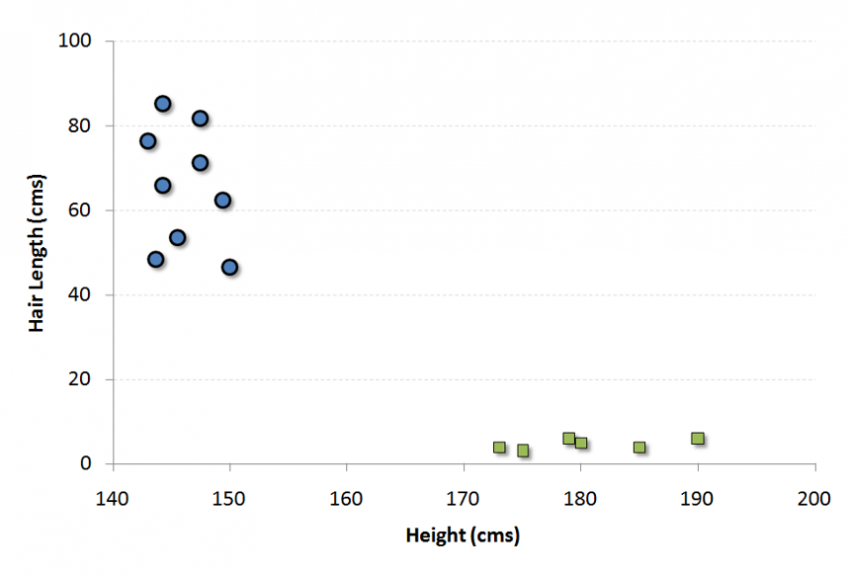


Fig 4. two different categories

Presently, we will discover some line that parts the information between the two diversely characterized gatherings of information. This will be the line with the end goal that the good ways from the nearest point in every one of the two gatherings will be farthest away.

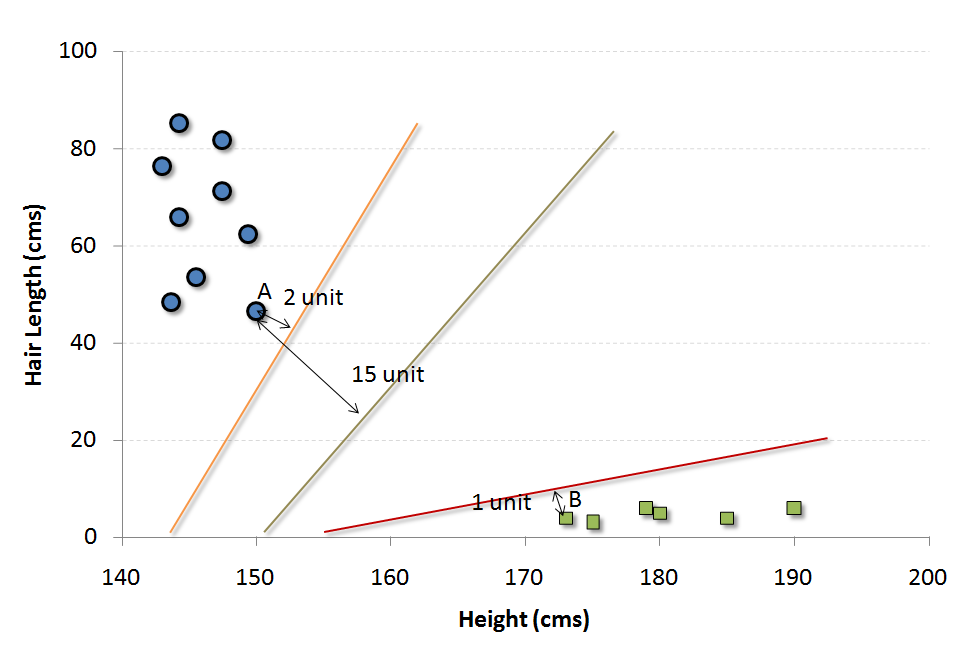


Fig 5. example of SVM [3]

In the model appeared over, the line which parts the information into two diversely characterized bunches is the dark line, since the two nearest focuses are the farthest separated from the line. This line is our classifier. At that point, contingent upon where the testing information arrives on either side of the line, that is the thing that class we can group the new information as.

**AdaBoostClassifier**

Bagging is an equal troupe in light of the fact that each model is manufactured autonomously. Then again, boosting is a successive gathering where each model is assembled dependent on rectifying the misclassifications of the past model.

Bagging generally includes 'basic democratic', where every classifier votes to acquire an ultimate result one that is dictated by most of the equal models; boosting includes 'weighted democratic', where every classifier votes to get an ultimate result which is controlled by the lion's share however the successive models were worked by allocating more prominent loads to misclassified occurrences of the past models.

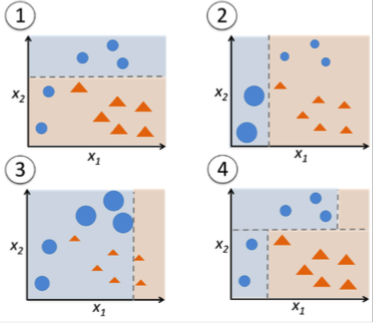


Fig 6. Adaboost for a decision tree. [4]

In contrast to bagging, you use very simple classifiers as base classifiers, so-called “weak learners.” Picture these weak learners as “decision tree stumps” – decision trees with only 1 splitting rule. Above, we will refer to the probably most popular example of boosting, AdaBoost. Here, we start with one decision tree stump (1) and “focus” on the samples it got wrong. In the next round, we train another decision tree stump that attempts to get these samples right (2); we achieve this by putting a larger weight on these training samples. Again, this 2nd classifier will likely get some other samples wrong, so you’d re-adjust the weights [4]

In a nutshell, we can summarize “Adaboost” as “adaptive” or “incremental” learning from mistakes. Eventually, we will come up with a model that has a lower bias than an individual decision tree (thus, it is less likely to underfit the training data).

**DecisionTreeClassifier**

This is one of my number one calculations and I use it regularly. It is a kind of managed learning calculation that is generally utilized for order issues. Shockingly, it works for both straight out and persistent ward factors. In this calculation, we split the populace into at least two homogeneous sets. This is done dependent on most huge ascribes/autonomous factors to make as unmistakable gatherings as could reasonably be expected.

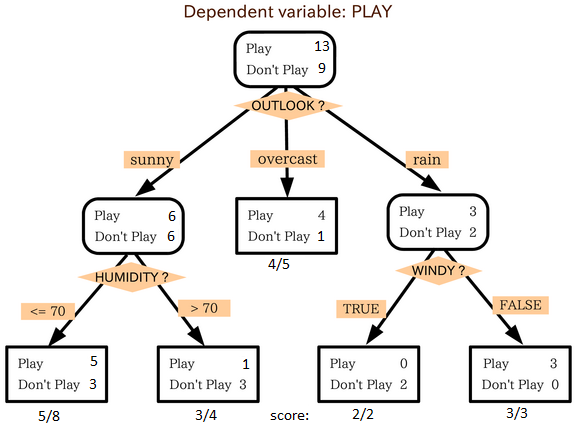


Fig 7. Example of DecisionTree Classifier

In the picture above, you can see that populace is arranged into four unique gatherings dependent on various ascribes to recognize 'on the off chance that they will play or not'. To part the populace into various heterogeneous gatherings, it utilizes different methods like Gini, Information Gain, Chi-square, entropy.

**Ensemble Learning**

A Voting Classifier is an ML model that trains on a troupe of various models and predicts a yield (class) in light of their most noteworthy likelihood of picking a class as the yield.

It just totals the discoveries of every classifier passed into the Voting Classifier and predicts the yield class dependent on the most elevated dominant part of casting a ballot. The thought is as opposed to making separate devoted models and finding the precision for every them, we make a solitary model which trains by these models and predicts yield dependent on their consolidated lion's share of deciding in favor of each yield class.

Voting Classifier upholds two sorts of votings.

* Hard Voting: In hard voting, the anticipated yield class is a class with the most noteworthy greater part of votes i.e the class which had the most elevated likelihood of being anticipated by every one of the classifiers. Assume three classifiers anticipated the yield class(A, A, B), so here the dominant part anticipated An as yield. Consequently A will be the last forecast.
* Soft Voting: In delicate voting, the yield class is the expectation dependent on the normal likelihood given to that class. Assume given some contribution to three models, the forecast likelihood for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the normal for class An is 0.4333 and B is 0.3067, the victor is plainly class An on the grounds that it had the most noteworthy likelihood found the middle value of by every classifier.

1. Evaluation

We briefly explain the metrics which were used to obtain the classification results.

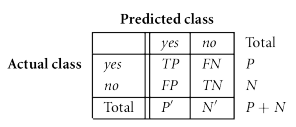
1) True Positive (TP): It is defined as the number of instances which are considered positive and are in fact positive.

2) True Negative (TN): It is defined as the number of instances which are considered negative and are in fact negative.

3) False Positive (FP): It is defined as the number of instances which are considered positive but are in fact negative.

4) False Negative (FN): It is defined as the number of instances which are considered negative but are in fact positive.

5) Confusion Matrix: It is a two dimensional square matrix with a row and column for each class. Each matrix element shows the number of instances for which the actual class is the row and the predicted class is the column.



6) Accuracy: Accuracy is defined as the sum of the number of true positives and true negatives divided by the total number of instances.



7) Precision: Precision is the ratio of true positive instances to the total number of positive instances which may include instances that are falsely considered positive.



8) Recall: Recall is the ratio of the number of true positive instances to the total numbers of instances that are in fact positive.



IV DATASETS

The dataset we used is manually collected begin and malware APKs from Android Malware Dataset (CIC-AndMal2017) [5]. The CICAndMal2017 dataset contains 10,854 samples (4,354 malware and 6,500 benign) from several sources. From these APKs, we chose 114 benign and 178 malware APKs due to computation limits.

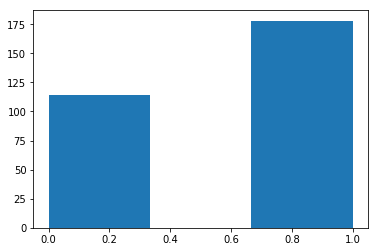


Fig 8. Plot of 114 benign and 178 malware APKs

V EXPERIMENTS AND RESULTS

Experiment evaluation is one of the key roles in machine learning. It tells us how well the model would work on production. Machine Learning Engineers choose the best model which is suitable for the business needs. We have trained the various machine learning classifiers with 70% data and 30% testing data. We splitted the data matrix into training and testing set to check the robustness of the model. After training the model with training data and testing with the test data, we got the following report of various machine learning models.

In the SVM Classifier, we got the confusion matrix as follows.

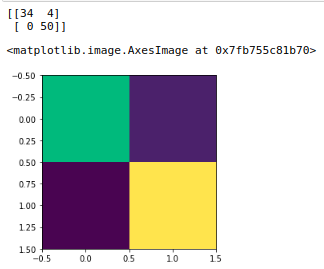


Fig. 9 Confusion Matrix of SVM

TP = 34 FN = 4

FP = 0 TN = 50

The above confusion matrix of SVM tells us that out of 38 positive (malign) APKs, they have correctly detected 34 APKs. However 4 APKs are misclassified as negative (benign) but they are maligned APKs. The same way there were 50 negative (benign) APKs and all are correctly classified.

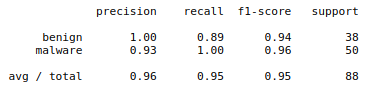


Fig 10. Report of SVM Classifier

accuracy = TP + TN / (TP + FP + TN + FN)

accuracy = 0.9545

precision = TP / (TP + FP)

precision = 1.0

recall = TP / (TP + FN)

recall = 0.89

In the Adaboost Classifier, we got the confusion matrix as below.

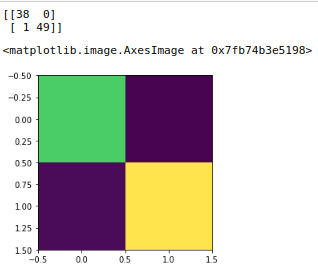


Fig. 11 Confusion Matrix of AdaBoost

TP = 38 FN = 0

FP = 1 TN = 49

The above confusion matrix of Adaboost classifier tells us that out of 38 positive (malign) APKs, they have correctly classified all the APKs. But in a negative scenario, out of 50 APKs, they misclassified 1 APK, which is actually negative (benign) but classified as a positive (malign).

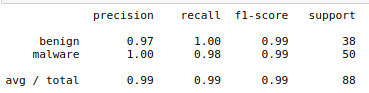


Fig. 12 Report of AdaBoost Classifier

accuracy = TP + TN / (TP + FP + TN + FN)

accuracy = 0.98

precision = TP / (TP + FP)

precision = 0.97

recall = TP / (TP + FN)

recall = 1.0

In the DecisionTree Classifier, we got the following confusion matrix.

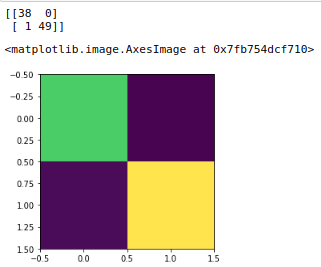


Fig. 13 Confusion Matrix of DecisionTree Classifier

TP = 38 FN = 0

FP = 1 TN = 49

The above confusion matrix of DecisionTree Classifier is somehow similar to the Adaboost one which tells us that out of 38 positive (malign) APKs, they have correctly classified all the APKs. But in a negative scenario, out of 50 APKs, they misclassified 1 APK, which is actually negative (benign) but classified as a positive (malign).

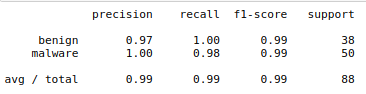


Fig. 12 Report of DecisionTree Classifier

accuracy = TP + TN / (TP + FP + TN + FN)

accuracy = 0.98

precision = TP / (TP + FP)

precision = 0.97

recall = TP / (TP + FN)

recall = 1.0

VI DISCUSSIONS

The approach of extracting the features out of APKs are somehow lengthy task. Hence, optimizing the data pipeline of getting features out of APKs is work open for research. After the extraction of the feature we read in the official guidelines of android development toolkit that they are continuously improving their SDK and APIs from time to time. Hence, there may be chances that the features we take in count to train machine learning gets deprecated and we have to reevaluate the pipeline from feature engineering to evaluation.

VIII CONCLUSION

ACKNOWLEDGEMENTS

I would like to acknowledge my indebtedness and render my warmest thanks to my supervisor, Professor Xiaoqin Zhang (University of Massachusetts Dartmouth), who made this work possible. Her friendly guidance and expert advice have been invaluable throughout all stages of the work.

REFERENCES

[1] K. Lab, “It threat evolution q3 2019. statistics,” 2019. [Online]. Available: https://securelist.com/it-threat-evolution-q3-2019-statistics/95269/

[2] Group-IB, “Group-ib uncovers android trojan named gustuff capable

of targeting more than 100 global banking apps, cryptocurrency

and marketplace applications,” 2019. [Online]. Available: https:

//[www.group-ib.com/media/gustuff/](http://www.group-ib.com/media/gustuff/)

[3] <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>

[4]https://sebastianraschka.com/faq/docs/bagging-boosting-rf.html

[5] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani, “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification”, In the proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST), Montreal, Quebec, Canada, 2018.