Master's Project Report

Android Malware Detection using Static and Dynamic Features

- Kaustubh Patro

01819140

Advisor

Dr.Hua Fang

Department of Computer and Information Sciences

University of Massachusetts Dartmouth

Android Malware Detection using Static and Dynamic Features

Kaustubh Patro
Department of Computer and Information Science
University of Massachusetts Dartmouth

Advisor
Dr. Hua Fang
Department of Computer and Information Science
University of Massachusetts Dartmouth

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 Decision Tree Classifier.**

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

I. 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. Most 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 organization information bundles or in some cases even a straightforward login interface.

Considering Trend Micro™ Mobile App Reputation information, 80% of the 1 million

applications are malevolent in nature [7]. 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 [7]. Thus, any android telephone over the web is on high danger and has the danger to their protection and security. Static and Dynamic [8] 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.

II. 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 [6] 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 [9], have proposed a static analysis approach dependent on consents from show document, also, broke down 37 most huge authorizations utilizing Chi-square include choice strategy, and characterized utilizing Naive Bayes classifier.

Ansul Arora et al.[10] 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 [11] proposed Mmda dependent on metadata in show record. They have extricated authorizations, all out number of authorizations utilized by one example, equipment highlights 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.[12] 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. [13] 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[14] 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[15] has proposed android malware identification utilizing upgraded arbitrary backwoods classifiers and PSO what's more, GA(Genetic Algorithm). They have separated consent sets from show documents 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 [16]	Multilevel data pruning to optimize permissions	Does not work on skewed dataset Optimized to 22 permissions	91.34%
Xiang Li[9]	- Permissions are extracted and ranked using chiSquare test.	- Optimized to 37 permissions	82.48%
Mmda [11]	- Permission sets, no of permissions, Hardware features, Receiver action, No of Receiver actions	Optimized to 122 features	94%
Sayfullina [13]	PERM, STR, DEX, HASH	Too large feature vector	99.91%
Sujithra[13]	Permissions used as features.	IG, KNN, PSO are used for optimization.	88.4% Using PSO
Arora [10]	Network traffics are	Find the optimum	87.25% using 22

	captured using a sniffer	number of features. - Cannot detect encrypted traffic, obfuscated malware	features
ScanMe [14]	cloud based sandbox perform Static and dynamic analysis	Processing and power capacity of device is not used. - Network bottleneck	86%

III. 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.

2. 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..

	A	B	C	D	E	F	G	H	I	
1	appname	lseek	clock_gettime	clone	close	connect	dup	epoll_ctl	epoll_pwait	fa
2	a35937acc1a17eeff9a2015c6bbafddaaac44897	0	113	1	6	0	2	1	27	
3	82552e838199f9a2d4a537588fb77ecbc44a3ea8	0	101	1	6	0	2	1	26	
4	730fed46dc7f136919061f46111ee5e05aa1b854e	0	5916	9	431	0	94	94	1501	
5	4de0d899794926 5a4b5647bb9f9d42926bd88191	0	5963	12	416	0	85	86	1492	
6	8a15729f0f7fcdb68782fe78429a714225e58610	0	11436	10	1011	0	50	41	2238	
7	b999d381d96cc3b40edce7048543f8a7a3f0e79e	0	1543	1	97	1	13	9	289	
8	4edab972cc232a2525d6994760f0f71088707164	8055	8262	12	618	0	71	51	1614	
9	d4afaf56089d4560254df7202aef36c7e09e408e	0	1	0	0	0	0	0	0	
10	98057097b1810c2a08a4662d11da3f8fe1bc6ef0	22805	4084	7	321	0	35	28	736	
11	16706fe77bf2fa73ef2d85543ae1839772d429f9	0	1	0	0	0	0	0	0	
12	a1fa9de17c36f00fbbdffc9bfc3c858b9202f73	0	1	0	0	0	0	0	0	
13	400acf4279ada32b148a7371b14672e8bbdac39e	14330	1655	6	134	0	24	14	273	
14	0274a66cd43a39151c39b6c940cf99b459344e3a	0	4291	2	113	0	37	33	1365	
15	02d2e109d16d160f77a645f44314fedcdbcd6e18	0	6161	10	426	0	68	62	1308	
16	2c1ea998acb4e71242ca757b03b7dd3de1c4cbc	0	6373	10	467	0	73	71	1237	
17	2ac72e9b0c4cbf4e3aaaf0f3614e53dbce703bfa	0	880	8	81	0	7	5	131	
18	2b889919deec496060a717770c8660803e2f6bd3	0	259	8	34	0	11	2	81	
19	8fc445ba6e8ef561607a41fc83008f92890a026f	0	4037	12	227	0	56	55	967	
20	7cddf3742646e176db32f2ec9acb7e0f44584b4b	0	4901	41	714	0	131	129	1340	

Fig 1. Tabular dataset view of extracted features.

```
 'return-void': 84,
 'return-wide': 1,
 'sget': 18,
 'sget-boolean': 5,
 'sget-object': 338,
 'sget-wide': 3,
 'sput': 14,
 'sput-boolean': 17,
 'sput-object': 22,
 'sput-wide': 7,
 'sub-int': 2,
 'sub-long/2addr': 6,
 'throw': 9,
 'xor-int/2addr': 1},
'Package name': 'denp.zsexnmjp.jlpuazhsotnr',
'Panda': False,
'Permissions': ['android.permission.CHANGE_NETWORK_STATE',
 'android.permission.SEND_SMS',
 'android.permission.RECEIVE_BOOT_COMPLETED',
 'android.permission.READ_PHONE_STATE',
 'com.android.alarm.permission.SET_ALARM',
 'android.permission.SYSTEM_ALERT_WINDOW',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.WRITE_SMS',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.WAKE_LOCK',
 'android.permission.GET_TASKS',
 'android.permission.CHANGE_WIFI_STATE',
 'android.permission.RECEIVE_SMS',
 'android.permission.READ_CONTACTS',
 'android.permission.INTERNET',
 'android.permission.WRITE_CONTACTS',
 'android.permission.READ_SMS'],
```

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.

3. Extracting dynamic features from API

To extract the dynamic features of the APK, we have used virus total'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.

```
malware
073a2f2d51c7dc00eb21e27cb8fa80f3.apkAPK WITH JSON. CONTINUE...
ff6648303056c0f09f584a0d01607bad.apkAPK WITH JSON. CONTINUE...
23cccc38d7397b8dc4613d0a53d78d63.apkAPK WITH JSON. CONTINUE...
0a6bd6d44fc7865847c5cc521be5edd0.apkAPK WITH JSON. CONTINUE...
No report received. Waiting...
No report received. Waiting...
12fb8804389e7496ff5a56e2134ccf5e.apkAPK WITH JSON. CONTINUE...
b3392cd774f9ff834aeb742a7a4a40e5.apkAPK WITH JSON. CONTINUE...
5deb5969dd67e5021bafa07c6ccc1330.apkAPK WITH JSON. CONTINUE...
e9068f116991b2ee7dcd6f2a4ecdd141.apkAPK WITH JSON. CONTINUE...
No report received. Waiting...
```

Fig 3. Sending APKs for dynamic analysis

4. 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.

5. 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 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 optimum.

Hence, to check on which algorithm works best on our feature matrix, we will apply the topmost 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 facility.

For instance, if 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)

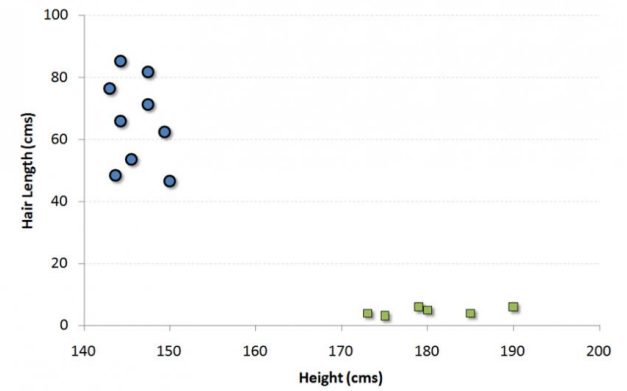

Fig 4. two different categories

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.

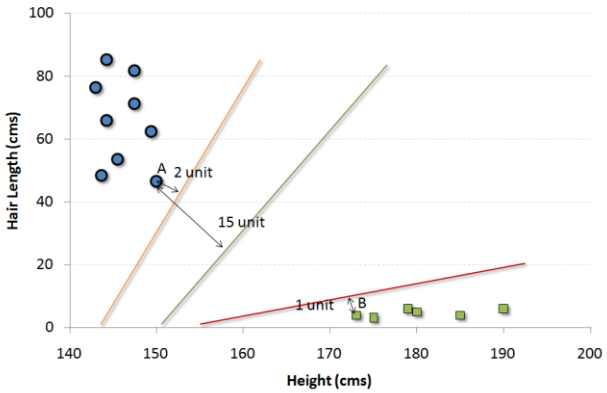

Fig 5. example of SVM [3]

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 because 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.

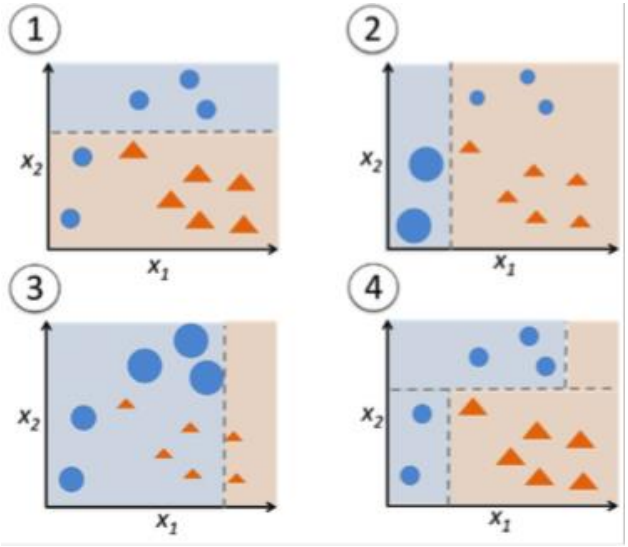

Fig 6. Adaboost for a decision tree. [4]

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).

Decision Tree Classifier

This is one of my number one calculation 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.

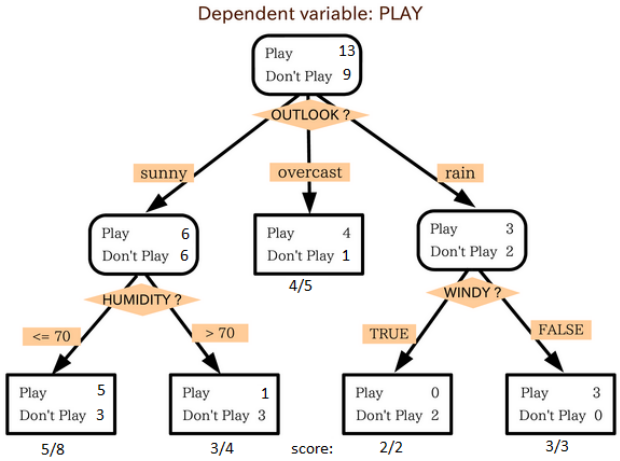

Fig 7. Example of Decision Tree Classifier

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) considering 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 voting's.

- 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 on the grounds that it had the most noteworthy likelihood found the middle value of by every classifier.

6. 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.

		Predicted class		
		yes	*no*	Total
Actual class	*yes*	*TP*	*FN*	*P*
	no	*FP*	*TN*	*N*
	Total	P'	N'	$P + N$

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

$$Accuracy = \frac{TP + FP}{TP + FP + TN + FN}$$

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.

$$Precision = \frac{TP}{TP + FP}$$

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

$$Recall = \frac{TP}{TP + FN}$$

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.

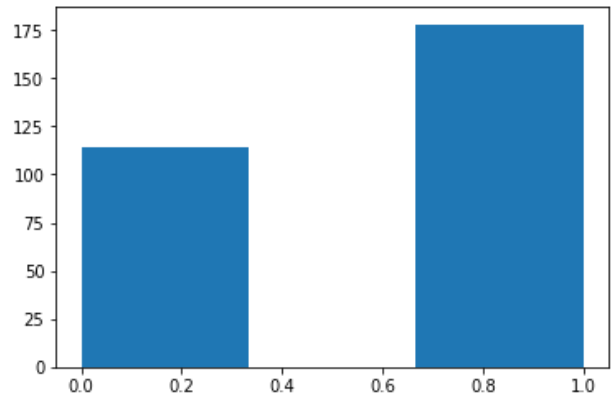

Fig 8. Plot of 114 benign and 178 malware APKs

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 split 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.

```
[[34  4]
 [ 0 50]]
```

```
<matplotlib.image.AxesImage at 0x7fb755c81b70>
```

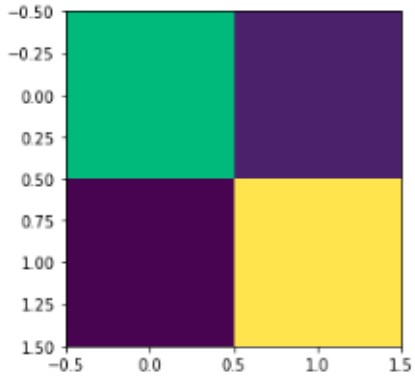

Fig. 9 Confusion Matrix of SVM

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.

	precision	recall	f1-score	support
benign	1.00	0.89	0.94	38
malware	0.93	1.00	0.96	50
avg / total	0.96	0.95	0.95	88

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.

```
[[38  0]
 [ 1 49]]

<matplotlib.image.AxesImage at 0x7fb74b3e5198>
```

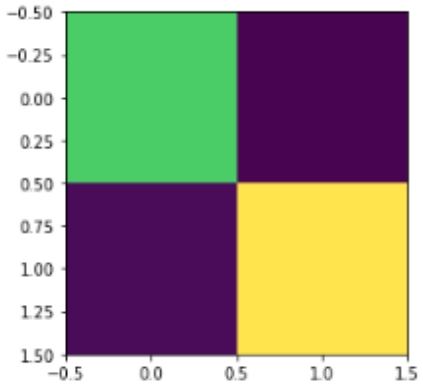

Fig. 11 Confusion Matrix of AdaBoost

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 negative (benign) but classified as a positive (malign).

	precision	recall	f1-score	support
benign	0.97	1.00	0.99	38
malware	1.00	0.98	0.99	50
avg / total	0.99	0.99	0.99	88

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 Decision Tree Classifier, we got the following confusion matrix.

```
[[38  0]
 [ 1 49]]

<matplotlib.image.AxesImage at 0x7fb754dcf710>
```

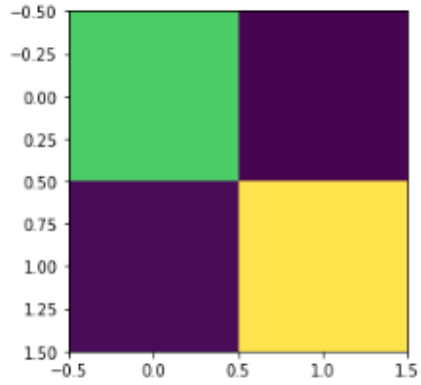

Fig. 13 Confusion Matrix of Decision Tree Classifier

TP = 38 FN = 0
FP = 1 TN = 49

The above confusion matrix of Decision Tree Classifier is somehow like 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 negative (benign) but classified as a positive (malign).

	precision	recall	f1-score	support
benign	0.97	1.00	0.99	38
malware	1.00	0.98	0.99	50
avg / total	0.99	0.99	0.99	88

Fig. 12 Report of Decision Tree 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 working methods of different machine learning models are unique in their way. Hence, we are getting different accuracies, precision and recall values for each algorithm. The crucial point we noticed that using ensemble learning algorithm such as adaboost and voting classifier may help you to get good state of the art accuracies but it may not work best for specific situation needs, We can use many models of classical machine learning but the main part of the machine learning expert to identify the one which works best for the situation needs like if there is one situation to detect malware APKs heavily and do not care about result on benign then we need to choose the model whose precision is best on positive cases.

VII FUTURE RESEARCH

The approach of extracting the features out of APKs are somehow lengthy tasks. 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. Here in this report, we mainly focused on how the various machine learning algorithms work differently on the same kind of data and which model we have to choose based on their precision and recall value. However, we think that using Deep Learning models may strengthen the detection power of models because of their deep architecture and robustness.

VIII CONCLUSION

Although ensemble methods can assist you with winning AI competition by formulating refined calculations and creating results with high accuracy, it is frequently not favored in the businesses where interpretability is more significant. Regardless, the adequacy of these techniques is unquestionable, and their advantages in proper applications can be gigantic. In fields, for example, medical care, even the littlest measure of progress in the exactness of ML calculations can be something important.

ACKNOWLEDGEMENTS

I would like to acknowledge my indebtedness and render my warmest thanks to my supervisor, Professor Hua Fang (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/

[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.

[6] L. Sun, Z. Li, Q. Yan, W. Srisa-an and Y. Pan, "SigPID: significant
permission identification for android malware detection," 2016 11th
International Conference on Malicious and Unwanted Software
(MALWARE), Fajardo, 2016, pp. 1-8.
[7]GoogleBouncer:
http://blog.trendmicro.com/trendlabs-securityintelligence/a-look-at-google-bouncer/
[8] [AndroidPRAGuardDataset]
http://pralab.diee.unica.it/en/AndroidPRAGuardDataset
[9] Li, X., Liu, J., Huo, Y., Zhang, R., & Yao, Y. (2016, August). An Android malware detection method based on AndroidManifest file. In Cloud Computing and Intelligence Systems (CCIS), 2016 4th International Conference on (pp. 239-243). IEEE.
[10] Arora, A., & Peddoju, S. K. (2017, January). Minimizing Network Traffic Features for Android Mobile Malware Detection. In Proceedings of the 18th International Conference on Distributed
Computing and Networking (p. 32). ACM.
[11] Wang, K., Song, T., & Liang, A. (2016, December). Mmda: Metadata Based Malware Detection on Android. In Computational Intelligence and Security (CIS), 2016 12th International Conference
on (pp. 598-602). IEEE.
[12] Alatwi, H. A. (2016). Android malware detection using categorybased machine learning classifiers. Rochester Institute of Technology
[13] Sayfullina, L., Eirola, E., Komashinsky, D., Palumbo, P., & Karhunen, J. (2016, December). Android Malware Detection: Building Useful Representations. In Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on (pp. 201-206). IEEE.
[14] Zhang, H., Cole, Y., Ge, L., Wei, S., Yu, W., Lu, C., ... & Pham, K. D. (2016). ScanMe mobile: a cloud-based Android malware analysis service. ACM SIGAPP Applied Computing Review, 16(1), 36-49
[15] Sujithra, M., and G. Padmavathi. "Research Article Enhanced Permission Based Malware Detection in Mobile Devices Using Optimized Random Forest Classifier with PSO-GA." (2016).
[16] L. Sun, Z. Li, Q. Yan, W. Srisa-an and Y. Pan, "SigPID: significant permission identification for android malware detection," 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, 2016, pp. 1-8.