Journal of Theoretical and Applied Information Technology
15th July 2023. Vol.101. No 13
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARATIVE STUDY FOR ANDRIOD MOBILE STATIC ANALYSIS ALGORITHMS

SARA MAHMOUD SHEHATA[1], **ISLAM HEGAZY**[2], **EL-SAYED M. EL-HORBATY**[3]

[1]Demonstrator, Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University, Eygpt

[2]Associate Professor, Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University,Egypt

[3]Professor of Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University, Egypt

E-mail: [1]sara.shehata@cis.asu.edu.eg, [2]islheg@cis.asu.edu.eg, [3]sayed.horbaty@outlook.com

ABSTRACT

Recently, there has been a rapid increase in the use of smartphones, several of which are connected to the internet. Because of the data movement, malware attacks have enormously increased. Malware causes unexpected behavior in smartphones such as strange charges on your phone bill, invasive adverts, contacts receiving strange messages, poor performance, appearance of new applications, abnormal data consumption and noticeable reduction in battery life. Nonetheless, smartphone users remain unprotected from malware attacks. Thus, mobile antivirus applications have been developed to overcome this issue. Since android has established itself as the industry's dominant operating system for smartphones, many antivirus applications are available in the android play store. This paper presents a comparative study of android mobile static analysis. Static analysis is used to classify malware android Apps through meta data file of APK. Furthermore, we used TF-IDF feature extractor and investigate algorithms for static analysis, such as decision tree, naïve bayes, random Forest, K-nearest neighbor, XGB, MLP, support vector machine, logistic regression, adaboost, ,lasso regression, ride regression , ANN and extra trees. We use two datasets small and large "Drebin". The results of small dataset show that Multi-layer perceptron (MLP) gives the best overall accuracy 98.84% but it takes the biggest execution time around 33.4 seconds and The results of large dataset show that Extra trees gives the best overall accuracy 99.48%.

Keywords:- *Mobile Security; Mobile Antivirus; Malware Analysis; Machine Learning; Classification*

1. INTRODUCTION

The use of smartphones is increasing due to their flexibility while providing as many services as a laptop. Many users prefer using smartphones and tablets instead of laptops [1]. The number of operating systems (OSes) for smartphones has increased with each mobile OS gaining a significant market share [2]. The most well-known mobile OSes are android, iOS, Windows, and Symbian. Some other mobile OSes are less used (BlackBerry, Samsung, etc.) [3]. android percent is 72.92% of devices as proved in the market share of mobile operating systems worldwide in October 2020, as shown in Figure 1 [4]. One factor in its success is how simple it is to create new applications and services for the android platform and post them to any android markets that are available: Google play store, Amazon app store, Samsung galaxy apps, etc [5].

Security is important because data movement has increased in the current generation [6]. Malware is any harmful program used to obstruct machine operation, collect private information, or get access to mobile devices used by individuals [7]. It is distinguished by its malicious aim, which goes against the requirements of the mobile user and excludes software that inadvertently harms users due to a flaw. Both genuine (malicious) malware and accidentally destructive software are frequently referred to as "badware.". The number of mobile malware is small compared to that of PCs. Nonetheless, we can expect malware for smartphones to evolve in the same trend as that of PCs. Hence, in the coming years, we will be faced with a large amount of smartphone malware [7].

Here, we investigate data mining, machine learning, and static analysis classifiers to identify dangerous android OS apps a priori without the requirement to download and run the application.

We study machine learning classification algorithms and compare between them according to overall accuracy and execution time.

The main contributions of this paper can be summarized in analyzing the efficiency and suitability of well-known ML classifiers to classify android malware.

We used term frequency-inverse document frequency (TF-IDF) feature extractor and investigate machine learning classifiers such as decision tree, naïve bayes, random Forest, K-nearest neighbor, XGB, MLP, support vector machine, logistic regression, adaboost, lasso regression, ride regression , ANN and our proposed solution extra trees. We use two datasets small and large “Drebin”. The results of small “Drebin” dataset show that Multi-layer perceptron (MLP) gives the best overall accuracy 98.84 but it takes the biggest execution time around 33.4 seconds.The results of large “Drebin” show that Extra Trees gives the best overall accuracy.

The rest of the paper is organized as follows. Section 2 presents mobile malware detection techniques. Section 3 discuss the background and related works. Section 4 illustrates malware types and analysis. comparative of static analysis in Section 5, Section 6 represents experiments, Section 7 summarizes research issues, Section 8 summarizes the conclusion and section 9 presents future works.

2. MOBILE MALWARE DETECTION TECHNIQUES

Smartphones can be scanned and cleaned of infections with antivirus software. Although there are numerous antivirus programs, their main function is to shield cellphones against infections and eliminate any viruses that are discovered. Techniques for detecting mobile malware include signature-based detection, specification-based malware detection, heuristic-based detection, anomaly-based detection also known as behavior-based detection, Sandbox detection, Data Mining and cloud based malware detection

2.1 Signature based detection: This is frequently found in old propagation. It is conventional antivirus software that scans all APK files and verifies them against the database of known viruses and malware types. It looks for any unusual behavior in the executable files that could indicate the presence of unidentified viruses [8]. Its drawbacks are it needs to be updated regularly and a sizable database is required to store it.

2.2 Specification-based malware detection: employs a rule set to determine how malicious the program is. The weakness of Specification-based malware detection was that it is frequently challenging to completely and accurately specify the entire set of acceptable behaviors a system should display.

2.3 Heuristic-based detection: employs heuristics by running applications or programs with suspect code within a runtime virtual environment. This circumvents the drawbacks of signature-based detection by preventing the vulnerable code from infecting the real-world setting. Instead of looking for well-known signatures, this sort of detection keeps track of keystrokes [8]. Its drawbacks are compared to signature-based systems, it generates a lot more false positives and additional code is required in addition to a third-party component.

2.4 Behavioral-based detection: The best method for dealing with new and anonymous infections is behavioral-based detection, which seeks out signs of compromise rather than the actual attack. Behavioral approaches identify executable behavior patterns at a higher degree of abstraction [8]. Its drawback is production of false positives.

2.5 Sandbox detection: With the usage of sandboxes, malicious code can be safely executed without endangering the host device or network.

2.6 Data mining detection: is one of the most recent trends in malware detection. “Data mining is the practice of analyzing large databases in order to generate new information”. Data mining aids in determining whether or not a program is malicious [2].

2.7 Cloud-based detection Applications from the Google Play Store are automatically checked for malware using a tool provided by Google called Bouncer. Bouncer inspects applications as soon as they are uploaded and compares them to known viruses [2].

3. BACK GROUND AND RELATED WORKS

Authors in [9], [10], [11], [12] approach the study of the efficacy of machine learning classifiers over a simple feature type (permission, intents, services).

Authors in [13] explore adversary models, such as the android HIV.
Authors in [14], DroidAPIMiner one of the earliest studies that combines APIs with package level and APIs information parameters. K-NN, linear SVM, and DT algorithms like ID3, C4.5 were evaluated on a dataset of 20,000 and 3,987 goodware and malware apps, respectively. K-NN performed with an accuracy of up to 99% while other classifiers achieved 96%.
In [15], authors coupled APIs with permissions and evaluated the effectiveness of SVM, artificial neural networks (ANN), and random forest (RF). They were using a dataset of 5,000 Goodware and 1,260 malicious programs. The authors assessed their strategy with up to 96% accuracy.
In [16], authors investigated the effectiveness of well-known classifiers on a dataset made up of 621 apps of goodware also 175 malicious apps. Results in this instance showed that combining permissions and API accuracy can be increased to a level of about 90.3%.
In [17], authors assessed the effectiveness of Deep Belief Network (DBN) on a dataset of 4,000 legitimate apps and 1,000 malicious ones. They showed through their investigation that DBN outperforms traditional ML classifiers and can achieve up to 93% F-Measure.The results of several well-known classifiers were integrated into an ensemble method to determine whether an app is malicious or not. The published findings for a dataset that included 445 goodware and 1,246 malware apps demonstrated a performance accuracy of up to 99%. The results of several well-known classifiers were integrated in an ensemble method to determine whether an app is malicious or not. The published findings for a data set that included 445 goodware and 1,246 malware apps demonstrated a performance accuracy of up to 99%.
In [18], authors showed the efficiency of a Bayesian Network (BN) was assessed using a dataset made up of 1,846 goodware and 5,560 malware apps. The results showed a True Positive Rate (TPR) of up to 95%. A very similar strategy was used in [19], where the authors showed accuracy of up to 99.7% after merging multiple features.
In [20], authors used a wide range of information as input to ML classifiers including intents, permissions, system commands, suspicious API calls, and other malicious actions (such accessing IMEI). The authors evaluated various classifiers (SVM, DT, RF, etc.) using a dataset that included 11,187 goodware and 18,677 malicious applications. The evaluated algorithms effectiveness was assessed using an F1-score that may reach up to 96%. A similar strategy was employed in [21], where the authors used the mix of APIs, commands, and permissions as a dataset. An accuracy of up to 95% was achieved during the evaluation of this method utilizing a dataset of 2,000 apps and a Bayesian classification system.
In [22], the accuracy of various ML classifiers, such as SVM, K-NN, Extra Randomized Tree (ERT), etc. Classifiers were assessed by using equal data partitions between malware and goodware. The authors use an extended set of features, such as API calls, permissions, etc., associated with 11,120 apps from the Drebin dataset.
A recent study [23] focuses on how factors affect how well ML classifications are made. Over SVM, the authors employ a number of feature selection techniques, including BI-Normal Separation and Mutual Information. According to the results, BI-Normal Separation chooses the optimal characteristics for reaching accuracy levels of up to 99.6%. In [24], authors present a multimodal malware detection method for android IoT devices employing a variety of features, with an emphasis on the optimizations of the NB classifier. The investigation shows that using such a method, accuracy can reach up to 98%.
In [25], the majority of methods focus on the effectiveness of well-known ML classifiers, such as K-nearest neighbour (K-NN), support vector machine (SVM), Naive Bayes (NB), decision tree (DT), etc., and differ on the features taken into account for classifying an app as malware.
The Drebin data set, which includes 123,453 goodware and 5,560 malware apps, is the basis for the analysis. Features from the Drebin dataset are present in the manifest file. Chi-square Test, Bernoulli Naive Bayes, Random Forest, L1 and L2 regularization, neural network, and Support Vector Machine were among the algorithms utilized.
In [26], authors determine how quickly classifiers expire; they examined the effects of idea drift on malware classifiers for android malware samples. They used four drift detectors, two representations (Word2Vec and TF-IDF), two classifiers (Adaptive Random Forest and Stochastic Gradient Descent classifier), and two representations (Word2Vec and TF-IDF) to analyse 480 K sample android apps from two datasets (DREBIN and AndroZoo) gathered over nine years (2009–2018). (DDM, EDDM, ADWIN, and KSWIN). For their best outcomes, Random Forest has an accuracy rate of 99.23%.

4. MALWARE TYPES AND ANALYSIS

4.1 Malware Types

Malware comes in many forms, however a virus is one of them and is more commonly known. Malware: malicious code that is specifically designed to disrupt, damage, or gain unauthorized access to a smart phone. Since malware can replicate itself and occasionally start an execution without any user assistance, its primary means of transmission is either user interaction or self-propagation. Malware that targets traditional workstations is currently more prevalent than malware that targets mobile devices. The fact that many businesses now permit employees to access corporate networks using personal devices presents an increasing hazard, though. The most prevalent types of malwares are viruses, trojans, adware, spyware, ransomware, worms, phishing, and pharming.

4.1.1 Virus: is a piece of computer code that can replicate itself and harm your device by altering your operating system or erasing data. 97% of mobile malware is installed on android devices.

4.1.2 Trojan: This type of malware joins a seemingly safe and reliable program or application. The Trojan is launched and infects the phone after legitimate the program or app has been installed. Malicious users may steal sensitive data, including card accounts or bank login information. In addition, it can be used to take over the browser, causing a smartphone to send premium messages without your permission. Additionally, it has the power to disable apps and disable your phone. Two of the most well-known phone Trojans are the Skulls and the Hummer.

4.1.3 Adware and Spyware: spyware masquerades as a legitimate program, mobile phone users are not aware that they have infected their device. This spyware stealthily gathers personal data, such as browsing history, messaging preferences, location, contacts, and downloads. The malware transmits information to a third party, such as an advertising or a marketing data company.

4.1.4 Ransomware: This kind of malware locks up a smartphone and demands a fee from its victims in order to restore control. More current ransomware can modify the PIN or security code and get access to the administrator rights of mobile devices. Users who access harmful and insecure websites or who download shady third-party program are more likely to become infected with this frightening malware.

4.1.5 Worms: Text messages sent as MMS and SMS can transmit this malware. Because this malware can be activated without human participation, it poses a risk. Worms spread throughout the devices. It might have hazardous and deceptive instructions. The Ikee is one of the most famous worms.

4.1.6 Phishing and Pharming: This malware switches users to dangerous websites from safe ones. It mimics a valid login or authentication page. When users enter their account or login information, nefarious parties may steal and use those credentials.

When utilizing an official store, such as Google play , the percent changes to 0.1. Signs that your mobile phone has malware include[27]: Unexpected charges may be a sign of a virus. Strange charges on your phone account. In order to make money, malicious programs make expensive phone calls or text messages. Additionally, since adware affects your phone with dangerous code, intrusive and intrusive advertisements are indications that you might have it installed. Strange text messages sent to contacts. Malware can infect a contact's device by using your mobile to send spam SMS. An infection may be confirmed by poor performance or a slowdown in performance. New programs. If new applications suddenly start to appear on your device, a malicious app can be downloading them. They might also include malware. Unusual data usage. Malicious program use the internet to send and receive information from their developers. A noticeable decrease in battery life is another sign. If your battery life is poor, you may be using a "bugged" or poorly written app. It might also indicate that your phone has a virus, though. What to do if you become infected. Put the smartphone in airplane mode first. This prevents any harmful apps from sending and receiving data. Check your most recent app installations next. If any have few downloads, persistently low ratings, and unfavorable reviews on the Google Play Store or the App Store, remove them. Any potentially harmful applications from your smartphone should be eliminated by this. Third, best practice advises that you should install antivirus software.

4.2 Malware Analysis

4.2.1 Static analysis: Static analysis entails looking at a program's source or, in certain circumstances, object codes without running it. In a typical static analysis approach, the analyzed app code is first represented as abstract models (such as call graphs, control-flow graphs, or class/sequence diagrams from the unified modelling language) for analysis. Static analysis has several benefits, including:

(a) It contains all of the source code and the application's manifest file, which is checked for data.
(b) It is faster than dynamic analysis. Compared to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

dynamic analysis, it utilizes less hardware (c). Numerous vulnerabilities, such as purpose injection, unauthorized access to private or protected resources, and private data leakage, can be found via static analysis. It is also used to detect test generation, clone detection, energy use, and permission abuse. The implementation issues with code verification and cryptography are then mentioned [28].

Manifest File Analysis: Names of attributes are matched with vulnerabilities in this phase. The top characteristics are listed below, along with indications of when each one is vulnerable.

Backup Flag: If Backup Flag's value is true, it's vulnerable. The program data, such as the application database and files, are copied to the SD card to create a backup. Application data stored on an SD card is less secure than data stored on a mobile device. This is so that written permission can be granted more easily for SD cards. If the data being backed up is not regarded as sensitive or hazardous, the vulnerability risk of enabling backup is evaluated to be Low.

Debug Flag: If its value is true, it is regarded as vulnerable. Debug mode is used by Android application developers to record crucial fields and check that they are accurate. But failing to switch the app to release mode makes it simple for hackers to sniff the app. Enabling a debug mode is thought to pose a high vulnerability risk because it can reveal a great deal of data while the application is running in this mode.

Exported Components: If (activities and services) exist, they are regarded as vulnerable. Note that opening up more components to other services and apps necessitates adding more interfaces to protect against attacks.

4.2.2 Dynamic analysis: Dynamic analysis, as opposed to static analysis, finds vulnerabilities that arise while a program is running. Due to the need for installing the apps and simulating user input, it is seen as being more complex than static analysis (i.e., inputs, touches, and clicks). There are numerous tests for dynamic analysis. Test generation, test effectiveness, and vulnerability coverage are some of their distinctions. Fuzz testing, concolic testing, and search-based testing are the most popular dynamic analytic tests. A computer program is subjected to "fuzz testing," an automated software testing approach, by being given erroneous, unexpected, or random data as inputs. When an error occurs, such as a crash, a failing built-in code assertion, or possibly a memory leak, the program is then watched for it.

Program variables are treated as symbolic variables during symbolic execution, whereas during concrete execution specific input pathways are tested. In search-based testing, a testing task is automated using meta-heuristic optimizing search techniques such simulated annealing and evolutionary algorithms [28].

4.2.3 Hybrid Analysis: This analysis combines static and dynamic features and is more accurate than either analysis alone [29].

5. COMPARSION OF STATIC ANALYSIS ALGORITHMS

We compare the same machine learning algorithms in small and large datasets. Algorithms used here are decision tree, naïve bayes, random Forest, K-nearest neighbor, XGB, MLP, support vector machine, logistic regression, adaboost, ,lasso regression, ride regression , ANN and our proposed solution extra trees.

5.1 Used Feature extraction:

Term Frequency-Inverse Document Frequency (TF-IDF):

It is widely used for word feature extraction. It can be summed up as determining how relevant a word is to a corpus or series of words in a text. The frequency of a term in the corpus offsets the meaning increase that occurs when a word appears more frequently in the text (dataset)

5.2 Used Algorithms:

5.2.1 Decision Tree: This is a collection of guidelines for categorizing data. It analyses the variables in a dataset, chooses the most crucial ones, and creates a tree of choices that best divides the data. Start at the root node and work your way up the classification tree until you reach the leaf (terminal) node. A DT simulates an if/else block of code, to put it briefly. Think about the trained DT shown in the figure (3) as an example.

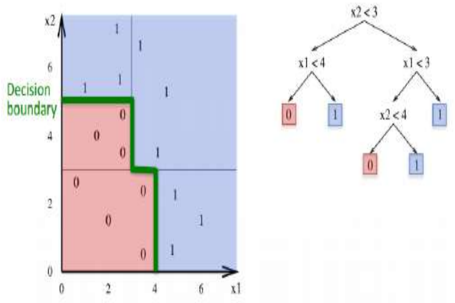

Figure 3:.How Decision Tree works

5.2.2 Naive Bayes: The Naive Bayes classifier operates under the presumption that the effects of each feature on a class are independent of one another. This presumption is known as class conditional independence[30].

$$P(\mathbf{h}|\mathbf{D}) = \frac{P(D|h)P(h)}{P(D)} \quad (1)$$

- P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.
- P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.
- P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.
- P(D|h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

5.2.3 Random Forest: is made up of a collection of decision trees. A decision tree (DT) divides the feature space into hyperboxes to solve a classification or regression problem. A decision tree's application produces a rough approximation of the decision boundaries when the decision surface is nonlinear. The growth of several trees guided by a variety of variables is a very popular method for stabilising the findings and enhancing the DT's forecasting ability. With this strategy, the idea of a forest of trees is introduced; the ultimate answer arises from averaging the individual answers. The Random Forests (RF) technique grows in parallel base learners, which must be statistically uncorrelated trees. RF is one of the most effective ensembles of trees and is based on the principle of uncorrelated trees. In order to accomplish this, feature subset selection during the training phase, which adds additional diversity to the learning of the decision trees, is a step that must be taken in addition to bootstrap aggregation (bagging) over the set of observations. The trees can offer useful information about the feature relevance because they divide the original feature space by design. Actually, the algorithm optimises a purity measure during training and chooses the feature (cost function) that increases the measure at each data split. the one that makes the tree less impure. By averaging the advancements each feature has made over all the trees in an ensemble of trees, the feature relevance is determined. The Gini index is one of the two purity metrics we examined.

Gini = 1 − ∑K k=1 P(Y ′ = k), Y ′ ⊆Y (2)

And the Entropy:

Entropy =

∑K k=1 − P(Y ′ = k)log2P(Y ′ = k), Y ′ ⊆Y (3)

5.2.4 K-nearest neighbour (KNN) is the most straightforward machine learning classifier. In contrast to other ML techniques, it does not result in a model.

It is a straightforward algorithm that classifies newly discovered examples using a similarity metric and stores all of the existing cases.

Assuming that our dataset has two variables a and b ,

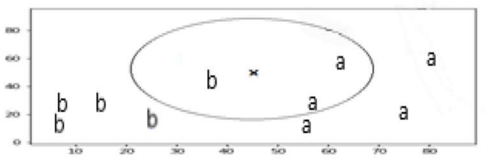

Figure 2: KNN example

KNN works by calculating the distances between a query and each example in the data, choosing the K examples that are closest to the query, and then voting for the label that is used the most frequently.

Advantages of KNN:

The algorithm is straightforward and simple to use.

There is no need for a model, parameter tuning, or additional presumptions.

Disadvantages of KNN:

As the number of cases and/or predictors/independent variables rises, the method becomes noticeably slower.

5.2.5 XGB: Extreme Gradient Boosting, often known as XGBoost, is a boosting technique based on gradient-boosted decision trees. One way that XGBoost differs from gradient boosting is by using a superior regularisation strategy to lessen overfitting [31].

Step 1: Calculate the similarity scores; this aids in the tree's growth.

Similarity Score is equal to (Sum of Remainders)2 / Remainders + Lambda

Step 2: calculate the decide to determine how to split the data.

Gain = Left tree (similarity score) + Right (similarity score)- Root (similarity score)

Step 3: Calculate the gain and gamma difference to prune the tree. (user-defined tree-complexity parameter)

Gamma gain

If the outcome is a positive number, do not prune; if it is a negative number, prune and once more deduct gamma from the subsequent Gain value up the tree.

Step 4**:** Calculate output value for the remaining leaves.

Output value = Sum of residuals / Number of residuals + lambda

5.2.6 MLP: A feed-forward neural network augmentation known as a multi-layer perceptron (MLP). It is made up of three different types of layers: the input layer, the output layer, and the hidden layer. The input layer is where the signal is received for processing at the input layer. The necessary tasks, such as classification and prediction, are finished by the output layer. The input and output layers are sandwiched between an arbitrary number of hidden layers that make up the MLP's actual computational engine [32].

5.2.7 SVM*:* This algorithm achieves notable accuracy while using minimal processing power. Finding a hyperplane in an N-dimensional space (N is the number of features) that clearly classifies the data points is the goal of the SVM method. Due of its sensitivity to noise, a small number of incorrectly labelled instances can significantly harm its performance..

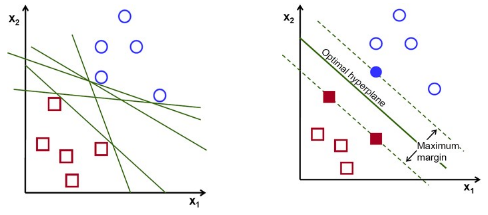

Figure 4: optimal hyperplane in svm.

5.2.8 Logistic Regression: This approach for classifying data uses discrete classes to group observations. Emails (spam or not spam), online transactions, and other types of data are some examples of classification issues (fraud or not fraud).returns a probability value after transforming its output using the logistic sigmoid function.

$$\hat{y} = 1/(1 + e^{\wedge}(-z)\,) \tag{4}$$

$$Z = W^{\wedge}T.x + b \tag{5}$$

Example: if we choose the threshold as 0.5 the prediction function ($\hat{y}$) returned a value of 0.7 then we would classify this observation as Class 1. If our prediction returned a value of 0.2 then we would classify the observation as Class 2.

5.2.9 Adaboost: Boosting techniques combine a number of low-accuracy classifiers to produce a classifier that is extremely accurate. Adaboost, a popular boosting algorithm, is an iterative ensemble method that creates a strong classifier by combining several poor classifiers to produce a strong classifier with high accuracy.

5.2.10 Lasso: adds a penalty for non-zero coefficients, but unlike ridge regression, which applies the so-called L2 penalty to the sum of squared coefficients, lasso applies the penalty to the sum of their absolute values (L1 penalty).

5.2.11 Ridge: penalizes the model based on the weights' sum squared value. Predictors' coefficients can be thought of as being set to zero when they are removed from the model. Let's penalize them if they deviate too much from zero instead of forcing them to be exactly zero, forcing them to be little continuously. In this manner, we maintain all of the model's variables while reducing model complexity.

5.2.12 Artificial Neural Network: are biologically inspired computer programs that replicate how the human brain processes information. Rather than using programming to learn, ANNs are trained through experience and learn by spotting patterns and relationships in data. The layers of the neural structure, also known as artificial neurons or processing elements (PE), are connected by coefficients and comprise hundreds of single units. (weights). The strength of brain calculations comes from the network of connections between neurons.

Proposed Solution :
Using extra trees classifier is one of the suggested solutions in Drebin dataset as no one used it before.

5.2.13 Extra Trees: The extra trees algorithm, like the random forests algorithm, creates a lot of decision trees, but the sampling of each tree is random and without replacement. As a result, a dataset with unique samples for each tree is produced. Additionally, a predetermined number of features from the entire set of features are randomly selected for each tree. The most important and distinctive feature of extra trees is the random selection of a splitting value for a feature. Rather than calculating a locally optimal split using Gini or entropy, the algorithm simply selects a split value at random. The result is a variety of unrelated trees.

$$\boldsymbol{Entropy(S) = \sum_{i=1}^{c} -piLog(pi)} \tag{6}$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

where c is the number of unique class labels and pi is the proportion of rows with output label is i.

5.3 Used Evaluation:

Calculate Overall Accuracy :

Overall Accuracy= $\frac{TP+TN}{TP+TN+FP+FN}$ **(7)**

- TP: True positives
- TN: True negatives
- FP: False positive
- FN: False negatives
-

Precision: Appropriate when minimizing false positives **is** the **focus.**

Precision= $\frac{TP}{TP+FP}$ (8)

Recall: Appropriate when minimizing false negatives is the focus.

Recall = $\frac{TP}{TP+FN}$ (9)

F1 Score:

F1 Score=2* $\frac{Precision * Recall}{Precision + Recall}$ (10)

6. EXPERIMENTS

6.1 Experiment 1: small dataset

android Malware Dataset for Machine Learning [19]: "Dataset consisting of feature vectors of 215 attributes extracted from 15,036 applications (5,560 malware apps from Drebin dataset and 9,476 benign apps). The dataset has been used to develop and evaluate multilevel classifier fusion approach for android malware detection, published in the IEEE Transactions on Cybernetics paper 'DroidFusion: A Novel Multilevel Classifier Fusion Approach for android Malware Detection. The description of the feature vectors/attributes obtained via static code analysis of the android apps."**.** Dataset Date is ''3-2021,'' .datasetLink:https://www.kaggle.com/datasets/shashwatwork/android-malware-dataset-for-machine-learning. We separate the data set to train 80% and test 20%. Used Features are manifest permission 53%, API call signature 33% and Other (30) 14%.

These features have been split in different groups: "hardware components (S1), requested permissions (S2), app components ((a) receivers, (b) activities, (c) content providers) (S3), filtered intents (S4), restricted API calls (S5), used permissions (S6), suspicious APIs (S7), and network addresses (S8)." [19].

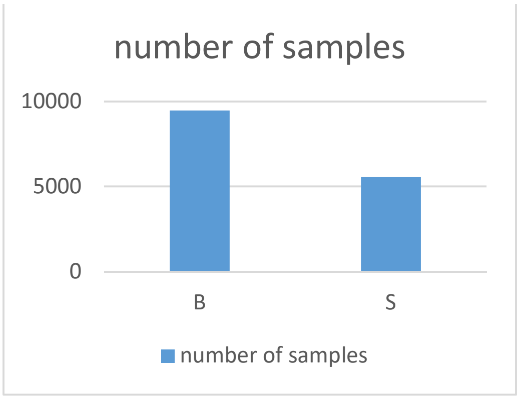

Figure 5: B samples of goodware, S samples of malware

Result:

Table 1

Algorithm Name	Accuracy %	F1Score	Recall	Precision	Execution Time (Seconds)
Decision Tree	97.54	98.17	98.06	98.27	0.237
Naïve bayes	70.88	69.52	53.62	98.81	0.235
Random Forest	98.60	98.99	**99.52**	98.46	3.935
K Neighbors	97.87	98.28	98.55	98.02	1.602
XGB	98.73	98.10	98.44	97.76	12.291
MLP	**98.84**	99.00	99.19	**98.82**	33.426
SVM	97.74	98.04	98.49	97.60	3.784
Logistic Regression	97.47	97.97	98.55	97.4	0.730
Adaboost	97.54	97.97	98.55	97.4	7.162
RidgeReg	80.84	81.2	81.68	80.72	**0.15**
LassReg	69.65	69.93	70.04	69.99	0.71
ANN	97.57	97.96	98.12	98.04	34.022
Extra Trees	98.80	**99.01**	99.46	98.56	2.085

After implementing machine learning classifiers on dataset and calculating the accuracy and execution time of each classifer.Multi layer perceptron (MLP) gives the best results but it takes the biggest execution time. Naive bayes gives the lowest accuracy.

6.2 Experiment 2: Large dataset

Android Malware Dataset for Machine Learning contains 123,453 benign and 5,560 malicious android applications. Dataset date is''8-2022'' .

Dataset Link is Fast & Furious: Malware Detection Data Stream | Kaggle. We separate the data set to train 80% and test 20%. Used Feature extractor: TF-IDF (using 120 features for each textual attribute).

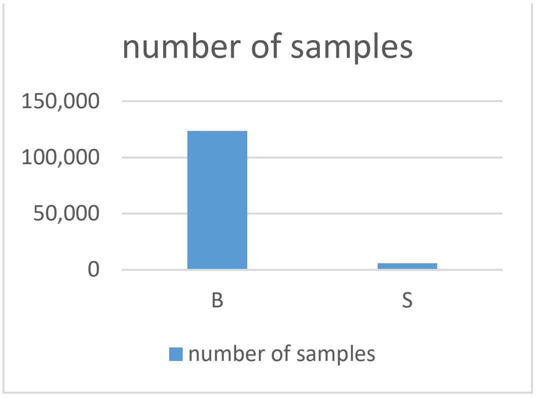

Figure 6: B samples of goodware, S samples of malware

Result:

Table 2

Algorithm Name	OverAll Accuracy %	F1Score	Recall	Precision	Execution Time (Seconds)
Decision Tree	99.15	90.59	89.2	90.2	348
Naïve bayes	25.43	69.5	53.6	98.8	**4**
Random Forest	99.46	93.62	90.33	97.16	124
K Neighbors	99.24	91.29	89.89	92.74	210
XGB	99.28	91.62	86.6	95	2732
MLP	99.29	91.8	89.89	93.85	541
SVM	98.76	85.18	80.31	90.67	888
Logistic Regression	98.8	85.54	80.57	91.15	20
Adaboost	98.68	84.11	79.08	89.82	1085
RidgeReg	62.8	54.37	51.21	57.94	5
LassReg	54.36	47.06	44.33	53.2	11
ANN	99.17	91.55	89.54	90.53	195
Extra Trees	**99.48**	**93.95**	**90.77**	**97.36**	295

Extra trees gives the best accuracy, F1Score, Recall and Precision.

Our proposed solution to use extra trees was effective in large "Drebin" dataset.

6.3 Difference from Prior Research

The best accuracy of the recent research on large "Drebin" dataset using random forest. Our proposed classifier extra trees was effective in large "Drebin" dataset. Extra Trees was better than Random Forest in large "Drebin" dataset

Advantages of Extra Trees over Random Forest:

Computational Efficiency: Extra Trees can be faster to train and evaluate compared to Random Forests since it avoids the computation of optimal splits at each node. This advantage can be significant, especially for large datasets or when time constraints are a concern.

Increased Randomness: Extra Trees introduces more randomness by randomly selecting feature thresholds for splitting, which can lead to further diversification of the trees. This increased randomness may help reduce overfitting and improve generalization performance, especially when dealing with noisy or high-dimensional datasets.

Robustness to Outliers: The random feature threshold selection in Extra Trees makes it more robust to outliers compared to Random Forests, as outliers are less likely to have a significant impact on the split decisions.

The following table show the comparison between our best results and the recent research in this field [26] best results We have better overall accuracy, F1Score, Recall, Precision..

Table 3

	Our study	The recent research
Overall accuracy	**99.48**	99.23
F1Score	**93.95**	90.63
Recall	**90.77**	85.85
Precision	**97.36**	96.31

7. RESEARCH ISSUES

The "Drebin" dataset is a widely used dataset in the field of Android malware research. It consists of a large collection of Android applications, both malicious and benign, and has been instrumental in developing machine-learning models for malware detection and classification.

While the "Drebin" dataset has been valuable for research purposes, it does have some limitations and potential issues that we should be aware of. Here are a few key points to consider:

8.1 Data Bias: The Drebin dataset was collected over a specific period and may not represent the current Android malware landscape. It might not include the latest malware variants or evolving techniques used by attackers.

8.2 Label Accuracy: The labeling process of the Drebin dataset involved using a combination of static analysis and manual labeling. Although efforts were made to ensure accuracy, there is still a possibility of mislabeled samples.

8.3 Limited Sample Size: While the Drebin dataset contains a substantial number of samples, it may still be considered small in comparison to the vast number of Android applications available. This limitation could impact the generalizability of research findings and the effectiveness of models trained solely on the Drebin dataset.

8. CONCLUSION

Android malware refers to malicious software or applications specifically designed to target Android devices. This malware can perform various harmful activities without the user's knowledge or consent, posing a threat to data security, privacy, and device functionality. Android devices account for 97% of mobile malware carriers. However, the percentage of malware becomes 0.1% when using an official store, such as Google play store. Our objective is to find an accurate and fast way to detect android. Static analysis has shown its robustness by preventing the installation of malware apps. Machine learning classifiers is fast and accurate way to detect malware from metadata (manifest file) of mobile application. We used the same feature extractor and classifiers in two dataset small and large "Drebin". We concluded that classifier accuracy increases with the increase of training data. Our proposed classifier extra trees was effective in large "Drebin" dataset. Feature extractor contributes to an increase in accuracy.

9. FUTURE WORK

Future work is based on research issues. Using different dataset that include :

10.1 The latest malware variants or evolving techniques used by attackers. We should be cautious about generalizing findings based solely on the Drebin dataset.

10.2 Unlabeling Samples: We should exercise caution when relying solely on the provided labels and consider additional validation measures.

10.3 Large Number of Samples: cover the vast number of Android applications available.

REFERENCES

[1] Available online at https://blog.globalwebindex.com/trends/device-usage-2019/, "How Device Usage Changed in 2018 and What it Means for 2019," last seen 28/12/2020.

[2] La Polla M, Martinelli F, and Sgandurra D, "A Survey on Security for Mobile Devices," In IEEE Communications Surveys and Tutorials, vol. 15, no. 1, pp. 446-471, 2013, DOI: 10.1109/SURV.2012.013012.00028

[3] Hamed T, Dara R, and Kremer SC, "Intrusion Detection in Contemporary Environments," In Computer and Information Security Handbook, 2017

[4] Available online at https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/#:~:text=Market%20share%20of%20mobile%20operating%20systems%20worldwide%202012%2D2020&text=Android%20maintained%20its%20position%20as,of%20the%20global%20 market%20share, "Market Share of Mobile Operating Systems Worldwide 2012-2020," last seen 28/12/2020.

[5] Martín I, Hernández JA, Muñoz A, and Guzmán A, "Android Malware Characterization Using Metadata and Machine Learning Techniques," Security and Communication Networks, vol. 2018, pp. 1-11, 2018.

[6] Rad BB, Masrom M, and Ibrahim S, "Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey," IJCSI International Journal of Computer Science Issues, vol. 8, no. 1, 2011.

[7] Talukder S and Talukder Z, "A Survey on Malware Detection and Analysis Tools," International Journal of Network Security and Its Applications, vol. 12, no. 2, pp. 12. DO - 10.5121/ijnsa.2020.12203.

[8] Riasat R, Sakeena M, Chong WA, Sadiq AH, and Wang YJ, "A Survey on Android Malware Detection Techniques," DEStech Transactions on Computer Science and Engineering, 2017.

[9] Xiaoyan Z, Juan F, Xiujuan W (2014) Android malware detection based on permissions. In: International conference on information and communications technologies (ICT 2014) .

[10] Liu X, Liu J (2014) A two-layered permission-based android malware detection scheme. In: 2nd IEEE international conference on mobile

cloud computing, services, and engineering, pp 142–148 .

[11] Sun L, Li Z, Yan Q, Srisa-An W, Pan Y (2016) SigPID: significant permission identification for android malware detection. In: 11th International conference on malicious and unwanted software (MALWARE) .

[12]Tchakounté F, Wandala AD, Tiguiane Y (2019) Detection of android malware based on sequence alignment of permissions. Int J Comput (IJC) 35:26–36.

[13] Hou S, Saas A, Ye Y, Chen L (2016) DroidDelver: "an android malware detection system using deep belief network based on API call blocks". In: Web-age information management, WAIM, vol 9998. pp 54–66.

[14] Choi S, Sun K, Eom H ,"Android malware detection using library API call tracing and semantic-preserving signal processing techniques. Report"

[15] Vij D, Balachandran V, Thomas T, Surendran R (2020) GRAMAC: a graph based android malware classification mechanism. In: CODASPY '20: proceedings of the tenth ACM conference on data and application security and privacy, pp 156–158

[16] Kang B, Yerima SY, Sezer S, McLaughin K (2016) N-gram opcode analysis for android malware detection. Int J Cyber Situational Awareness 1(1):1–24

[17] Chen YM, Hsu CH, Chung KCK (2019) A novel preprocessing method for solving long sequence problem in android malware detection. In: Twelfth international conference on ubi-media computing (ubi-media), pp 12–17

[18] Wu S, Wang P, Li X, Zhang Y (2016) Effective detection of android malware based on the usage of data flow APIs and machine learning. Inf Softw Technol

[19] Lou S, Cheng S, Huang J, Jiang F (2019) TFDroid: android malware detection by topics and sensitive data flows using machine learning techniques. In: IEEE 2nd international conference on information and computer technologies (ICICT), pp 30–36

[20] Ma Z, Ge H, Liu Y, Zhao M, Ma J (2019) A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access 7:21235–21245

[21] Shan P, Li Q, Zhang P, Gu Y (2019) Malware detection method based on control flow analysis. In: ICIT 2019: proceedings of the 2019 7th international conference on information technology: IoT and Smart City, pp 158–164

[22] Rana MS, Gudla C, Sung AH. Evaluating machine learning models for android malware detection: A comparison study. In: Proceedings of the 2018 VII International Conference on Network, Communication and Computing, ICNCC 2018. New York, NY, USA: Association for Computing Machinery; 2018. p. 17–21. https://doi.org/10.1145/3301326.3301390.

[23] Singh AK, Jaidhar CD, Kumara MAA. Experimental analysis of android malware detection based on combinations of permissions and api-calls. Journal of Computer Virology and Hacking Techniques 2019;15(3):209–18. https://doi.org/10.1007/ s11416-019-00332-z.

[24] Kumar R, Zhang X, Wang W, Khan RU, Kumar J, Sharif A. A multimodal malware detection technique for android iot devices using various features. IEEE Access 2019;7:64411–30. https://doi.org/10.1109/ACCESS.2019.2916886.

[25] VasileiosSyrris and DimitrisGeneiatakis " On machine learning effectiveness for malware detection in Android OS using static analysis data", Journal of Information Security and Applications,Volume 59, June 2021, 102794.

[26] Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Felipe Pinagé, Luiz S. Oliveira, André Grégio,,"Fast & Furious: On the modelling of malware detection" as an evolving data stream, Expert Systems with Applications,Volume 212,2023.

[27] Available online at https://www.pensar.co.uk/blog/signs-your-phone-has-avirus," 7 Signs your Phone has a Virus," last seen 1/10/2020

[28] Amr Amin 1,†, Amgad Eldessouki 1,†, Menna Tullah Magdy 1,†, Nouran Abdeen 1,† , Hanan Hindy 1,2,* and Islam Hegazy," AndroShield: Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach", MDPI, Basel, Switzerland, 2019.

[29] Baskaran B and Ralescu A, "A Study of Android Malware Detection Techniques and Machine Learning", MAICS2016. pp. 15-23, 2016.

[30] available online at Naive Bayes Classifier Tutorial: with Python Scikit-learn - DataCamp," Naive Bayes Classification Tutorial using Scikit-learn",last seen 13/4/2022.

[31] available online at "XGBoost for Regression - GeeksforGeeks" ,last seen 12/4/2022.

[32]available online at Multilayer Perceptron - an overview | ScienceDirect Topics," Multilayer Perceptron",last seen 10/4/2022.