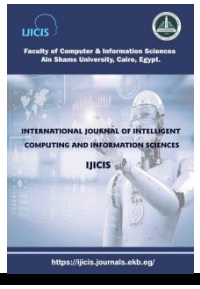
IJICIS, Vol.23, No.4, 19-32
DOI: 10.21608/ijicis.2023.242171.1300

International Journal of Intelligent Computing and Information Sciences

https://ijicis.journals.ekb.eg/



A STUDY FOR MALWARE STATIC ANALYSIS CLASSIFICATION ALGORITHMS WITH DIFFERENT FEATURES EXTRACTORS

Sara Mahmoud Shehata*

Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University, Cario, Egypt
sara.shehata@cis.asu.edu.eg

Islam Hegazy

Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University, Cario, Egypt
islheg@cis.asu.edu.eg

El-Sayed M. El Horbaty

Computer Science Department, Faculty of Computer and Information Sciences, Ain Shams University, Cario, Egypt
shorbaty@cis.asu.edu.eg

Abstract: *Smartphones are mobile devices that can connect to the Internet through various means such as Wi-Fi, cellular data networks (3G, 4G, 5G), or even through tethering to another device. Once connected to the Internet, smartphones can access a wide range of online services and applications, including web browsing, social media, email, streaming videos, online gaming, and much more. Malware attacks have significantly increased as a result of data movement. Malware causes unexpected smartphone behavior, including changing phone bill charges, intrusive advertisements, confusing messages being sent to contacts, unreliable performance, the appearance of new apps, unusual data use, and a noticeable drop in battery life. But smartphone consumers are still vulnerable to malware attacks. To solve this problem, we created a Malware detection system. Malicious Android Apps are categorized using static analysis through the APK's metadata file. "Drebin" dataset primarily uses the Android manifest file as one of the key features for Android malware detection. Additionally, we investigated algorithms for static analysis, including adaboost, ANN, decision trees, extra trees, K-nearest neighbors, lasso regression, logistic regression, MLP, naïve bayes, random forests, ride regression, support vector machines and XGB. We employ "Drebin" dataset with different feature extractors to reduce dataset dimensionality. We use TF-IDF and word2vec feature extractor. The experimental results show that TF_IDF performs better on "Drebin" dataset.*

Keywords: *Mobile Security; Feature Extractor; Malware Analysis; Machine Learning; Classification*
Received 2023-10-12; Revised 2023-11-27; Accepted 2023-12-13

1. Introduction

Google created the Android operating system for mobile devices [1]. It is most widely used in the world mobile operating system and is used on millions of devices, including smartphones, tablets, smart TVs, and other internet-enabled devices. Android is designed to be open source, which means that it is freely available for developers to use and modify. Android is also highly customizable and allowing users to

***Corresponding Author**: Sara Mahmoud Shehata

Computer Science Department, Faculty of Computer and Information Science, Ain Shams University, Cairo, Egypt

Email address: sara.shehata@cis.asu.edu.eg

change the look and feel of their device. One of the key features of Android is the Google Play Store, which is the official Android application store. The Play Store offers a vast selection of apps, games, movies, music, and books that users can download and install on their devices. Other notable features of Android include a built-in voice assistant called Google Assistant, support for multiple user profiles, and a variety of security features, including biometric authentication and regular security updates. Additionally, a variety of communication methods are supported by Android, including Wi-Fi, Bluetooth, NFC, and mobile data [1]. Android malware refers to any malicious software that targets Android devices. Malware can take many forms, and it can be distributed through various channels, including malicious apps, email attachments, phishing websites, and even SMS messages. Once installed on an Android device, malware can cause a range of problems, such as stealing sensitive information, showing intrusive advertisements, and seizing the device, or encrypting files and demanding a ransom for their release [2]. In this article, we investigated machine learning classifiers on "Drebin" dataset like including adaboost, ANN, decision trees, extra trees, K-nearest neighbors, lasso regression, logistic regression, MLP, naïve bayes, random forests, ride regression, support vector machines and XGB. We use two feature extractors Word2Vec and TF-IDF. The result shows that TF-IDF gives better accuracy.

The rest of the paper is organized as follows. Section 2, the background and related works are covered. Section 3 summarizes Android Malware Types and Analysis, Section 4 illustrates methodologies. Experiments are presented in Section 5 and Section 6 presents the paper stating the conclusion and future works.

2. Background and Related Works

In [3], DroidAPIMiner was one of the first studies to integrate APIs with package level and APIs information parameters. K-NN, linear SVM, and DT algorithms such as ID3, C4.5 were evaluated on a dataset comprising 20,000 and 3,987 goodware and malware apps, respectively. While K-NN achieved up to 99% accuracy, other classifiers only achieved 96% accuracy.
In [4], Using a dataset of 621 goodware and 175 malicious apps, authors investigated the efficacy of popular classifiers. Results in this instance showed that combining permissions and APIs can increase accuracy to a level of roughly 90.3%.
In [5], the authors showed how a dataset with 1,846 legitimate apps and 5,560 malicious ones could be used to assess a Bayesian Network's (BN) efficacy. The results showed a True Positive Rate (TPR) of up to 95%.
In [6], the accuracy of different Machine Learning (ML) classifiers, including Extra Randomized Tree (ERT), K-NN, SVM, and others. By splitting the data into equal portions for malware and goodware, classifiers were assessed. The Drebin dataset contains 11,120 apps, and the authors use a wide range of features, including permissions, API calls, and additional ones.
In [7], using a dataset comprising 1,000 malicious apps and 4,000 legitimate apps, the authors tested the Deep Belief Network (DBN). They found that DBN can achieve up to 93% F-Measure and performs better than conventional ML classifiers. An ensemble method was developed by combining the output of multiple popular classifiers to assess whether an app is malicious or not. Results for a dataset with 1,246 malware apps and 445 goodware apps were published. The dataset showed up to 99% performance accuracy. An ensemble method was used to combine the output of multiple popular classifiers to determine whether an app is malicious or not. Results for a data set with 1,246 malware apps and 445 goodware apps were published, and they showed up to 99% performance accuracy.

In [8], In order to illustrate the effectiveness of a Bayesian Network (BN), the authors used a dataset that included 5,560 malware apps and 1,846 goodware apps. The accuracy achieved by the authors after combining multiple features was up to 99.7%.

In [9], the authors fed suspicious API calls, intents, permissions, system commands, and other malicious actions (like IMEI access) into ML classifiers. Using a dataset of 11,187 legitimate software and 18,677 malicious programs, the authors tested a number of classifiers (SVM, DT, RF, and so forth). An F1-score with a maximum of 96% was used to evaluate the algorithms' efficacy. A dataset consisting of commands, permissions, and APIs was utilized by the authors in [10], who also used the same approach. This method was evaluated with up to 95% accuracy using a dataset of 2,000 apps and a Bayesian classification system.

A recent study [11] focuses on the factors that influence how well Machine Learning (ML) classifications are made. The authors use a variety of feature selection techniques, including BI-Normal Separation and Mutual Information, in addition to SVM. According to the findings, BI-Normal Separation selects the best characteristics to achieve accuracy levels of up to 99.6%.

In [12], with an emphasis on Naive Bayes (NB) classifier optimizations, the authors present a multimodal malware detection approach for Android IoT devices that makes use of a number of features. The results show that accuracy with this approach can be as high as 98%.

In [13], the authors evaluated the efficacy of SVM, ANN, and random forest (RF) and integrated APIs with permissions. A dataset comprising 1,260 malicious and 5,000 goodware applications was utilized by them. Up to 96% accuracy was achieved in evaluating the authors' approach.

In [14], while there are differences in the features taken into account when classifying an application as malware, most approaches center on the efficacy of popular machine learning (ML) classifiers like K-Nearest Neighbour (K-NN), Support Vector Machine (SVM), Naive Bayes (NB), Decision Tree (DT), and so forth. The Drebin data set, which comprises 5,560 malicious apps and 123,453 goodware apps, serves as the basis for the analysis. Elements from the Drebin dataset are included in the manifest file. The Chi-square Test, Random Forest, Bernoulli Naive Bayes, L1 and L2 regularization, neural network, and Support Vector Machine were some of the algorithms that were employed.

In [15], the authors examined how quickly malware classifiers expire and how concept drift affects malware classifiers for malware samples that are specific to Android. These employed two classifiers (Adaptive Random Forest and Stochastic Gradient Descent classifier), two representations (Word2Vec and TF-IDF), four drift detectors, and two representations (Word2Vec and TF-IDF) to examine 480 K sample Android apps from two datasets (DREBIN and AndroZoo) gathered over a nine-year period (2009-2018). Random Forest yields the best results with an accuracy rate of 99.23%.

In [16], the decision trees, naive bayes, random forests, K-nearest neighbors, XGB, MLP, support vector machines, logistic regression, adaboost, lasso regression, ride regression, artificial neural networks, and additional trees were among the static analysis techniques that the authors investigated. They used the small and large "Drebin" datasets. Extra trees offer the best overall accuracy with a large dataset, 99.48%, while Multi-layer perceptron (MLP) offers the best overall accuracy with a small dataset, but it has the longest execution time (33.4 seconds).

3. Android Malware Types and Analysis

In this section, we present the most common android malware types and android malware analysis.

3.1 Android Malware Types

Android malware refers to malicious software specifically designed to target devices running the Android operating system. As with other types of malware, Android malware aims to exploit vulnerabilities, steal information, gain unauthorized access, or cause harm to Android devices. Here are some common types of Android malware [17]:
Trojans: Once they are set up, trojans can carry out a number of nefarious tasks, such as stealing personal data, sending costly text messages, or allowing unauthorized users access to the device.
Ransomware: When a victim tries to access their data or device, ransomware prevents them from doing so until a ransom is paid.
Spyware: As it enables cybercriminals to obtain private and secret information, it is frequently used for espionage, identity theft, and other nefarious reasons.

3.2 Malware Analysis

The objective of Android malware analysis is to uncover the malware's behavior, characteristics, capabilities, and potential risks it poses to devices and users. This process involves various techniques, such as static analysis, dynamic analysis and hybrid analysis [18].

3.2.1 Static Analysis:

Static analysis of Android malware refers to the process of examining the code, resources, and other characteristics of an Android application (APK) without starting the application or executing it. This analysis is conducted to identify potential malicious behavior, security vulnerabilities, and other suspicious attributes within the APK file itself. It does not involve running the application on a device or emulator. In static analysis, security researchers, analysts, or automated tools analyze the structure, source code, permissions, manifest file, and other components of the APK to detect any indicators of malware, such as code that performs unauthorized or malicious actions, hardcoded malicious URLs, the use of sensitive permissions, hidden functionality, or attempts to obfuscate or hide malicious behavior.
The goal of static analysis is to uncover the inner workings of an Android application and identify any potential security threats or risks it may pose, allowing for the development of countermeasures or mitigation strategies to protect users and devices from the malware's harmful effects. Static analysis is an important step in the broader process of Android malware analysis and serves as an initial assessment of an APK's suspicious or malicious nature [16,19].

3.2.2 Dynamic Analysis:

Dynamic Android malware analysis refers to the process of analyzing and understanding the behavior of a suspicious or malicious Android application APK by running it in a controlled and monitored environment. This analysis involves executing the APK on a device or emulator while observing its interactions with the operating system, device resources, network, and other software components in real-time. The primary goal of dynamic analysis is to uncover the malware's actions, intentions, and potential harm it can cause when executed, without risking actual harm to the user's device or data [16,19].

3.2.3 Hybrid Analysis

Static and dynamic analysis are used to perform hybrid analysis, which looks at malware using both methods. Static analysis, for instance, can be used to spot possible dangers, while dynamic analysis can

be used to watch the behavior of the virus in real time. Because it gives a full understanding of both the code and behavior of a sample, hybrid analysis is frequently regarded as the most efficient way of malware analysis. [16].

4. Methodology

4.1 Dataset

The Drebin dataset is a well-known dataset used for Android malware detection research. It contains a collection of Android applications, both benign and malicious, that were gathered for the purpose of evaluating and training machine learning models to recognize malware. The Drebin dataset primarily focuses on Android malware, specifically the types of malwares that can infect Android devices. It includes various families and types of malwares that have been identified and analyzed by researchers. The dataset contains samples of malicious apps that exhibit a range of behaviors. The most common malware applications in “Drebin” dataset includes FakeInstaller, DroidKungFu, GoldDream and GingerMaster [20]
The goal of using the Drebin dataset is to develop and test machine learning models that can accurately distinguish between benign and malicious Android applications. These models can then be used in real-world scenarios to enhance mobile security and protect users from potentially harmful applications. We used Drebin Dataset for Machine Learning contains 5,560 malicious and 123,453 benign Android applications. Dataset upload date is ‘’8-2022” [21].

4.2 Feature Extractor

A feature extractor for text is a method or algorithm that converts raw text data into numerical or categorical features that can be used as input for machine learning models. Text data is typically unstructured, so feature extraction is essential to represent text in a format that can be processed by machine learning algorithms. We use two feature extractors Word2Vec and TF-IDF which are suitable for Drebin dataset [21]. These feature extractors convert textual data such as the content of Android app files in the "Drebin" dataset into numerical representations that machine learning classifiers can understand and process effectively.

4.2.1 Word2Vec

Word2Vec is an unsupervised learning algorithm that learns word embeddings from large amounts of text data. The basic idea behind Word2Vec is to represent each word in a continuous vector space, where words with similar meanings are closer together. Word embeddings capture semantic relationships between words, making them valuable for various natural language processing (NLP) tasks. It was introduced by researchers at Google in 2013 and has since become a fundamental tool in natural language [22].
Word2Vec works by training a neural network on a large corpus of text using either the Continuous Bag of Words (CBOW) or Skip-gram model. Both CBOW and Skip-gram have the same goal: to learn word embeddings, but they have different approaches to achieve it.

Continuous Bag of Words (CBOW):

In CBOW, the model predicts the target word based on the context words surrounding it. The context words are used as input, and the target word is the output. The architecture is shown in figure 1. The context words are represented as the encodings, and the hidden layer learns the word embeddings. The output layer predicts the target word probability distribution based on the context words. The word embeddings are updated during the training process to minimize the prediction error [22].

Figure 1: CBOW Architecture

Skip-gram:
In Skip-gram, given a target word, the model predicts context words. The target word is used as input, and the context words are the outputs. The architecture is shown in figure 2. The target word is represented as the encoding, and the hidden layer learns the word embeddings. During training, the model tries to maximize the probability of context words given the target word [22].

Figure 2: Skip-gram Architecture

4.2.2 TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency [23]. It is a numerical representation of text data that aims to highlight the importance of individual words in a collection of documents. TF-IDF is commonly used in information retrieval and text mining tasks, such as document similarity, document classification, and search engines. The TF-IDF score of a word in a document is calculated as the product of two components:
Term Frequency (TF): It measures how frequently a word appears in a document relative to the total number of words in that document. A word with a high TF score occurs more often in the document. The TF score is calculated as shown in equation 1.

$$\mathrm{TF} = \frac{\text{Number of occurrences of a word in the document}}{\text{Total number of words in the document}} \quad (1)$$

Inverse Document Frequency (IDF): It measures the rarity of a word across all documents in the dataset. Words that occur in many documents receive lower IDF scores, while words that appear in few documents get higher IDF scores. The IDF score is calculated as shown in equation 2.

$$\mathrm{IDF} = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the word}}\right) \quad (2)$$

The TF-IDF score is obtained by multiplying the TF and IDF scores for each word. This score gives higher importance to words that are frequent within a document (high TF) but rare across all documents (high IDF). The idea is to emphasize the uniqueness of words in specific documents and downplay common words that appear in many documents. The TF-IDF representation creates a sparse vector for

each document, where each element corresponds to the TF-IDF score of a specific word in the document. These vectors can be used as features in machine learning algorithms or for calculating document similarity scores.

Here's a simplified example of TF-IDF calculation for a single word in a document:

Consider a document: "Machine learning is an exciting field, and machine learning techniques are widely used."

Let's calculate the TF-IDF score for the word "machine" in this document:

1. Term Frequency (TF) of "machine": TF = Number of occurrences of "machine" / Total number of words in the document TF = 2 / 15 ≈ 0.133
2. Inverse Document Frequency (IDF) of "machine": Suppose there are 1,000 documents in the dataset, and "machine" appears in 200 of them. IDF = log(1000 / 200) ≈ 1.609
3. TF-IDF score of "machine": TF-IDF = TF * IDF ≈ 0.133 * 1.609 ≈ 0.214
   The TF-IDF score for "machine" in this document is approximately 0.214. The process is repeated for all words in the document to create the TF-IDF vector.

TF-IDF has some limitations that include:

1. It performs a straight word-count space computation of document similarity, which might be slow for big vocabularies.
2. It is predicated on the idea that word counts serve as independent proof of similarity.
3. There is no use of word semantic similarity in it.

4.3 Classifiers

Assigning each data point to a predetermined label or category is the main task that classifiers are employed for. The term "classes" refers to the set of potential labels [24]. Here we use supervised thirteen classifiers: adaboost, ANN, decision trees, extra trees, K-nearest neighbors, lasso regression, logistic regression, MLP, naïve bayes, random forests, ride regression, support vector machines and XGB.

4.3.1 AdaBoost (Adaptive Boosting):

AdaBoost is an ensemble method that integrates weak learners into a strong learner. It assigns higher weights to incorrectly classified instances in each iteration, forcing the model to focus on those instances [16].

4.3.2 Artificial Neural Network (ANN):

ANNs are versatile models inspired by the human brain's neural networks. They consist of input, hidden, and output layers, and they can learn complex patterns in data. Deep neural networks (DNNs) have many hidden layers and excel at feature extraction [25].

4.3.3 Decision Tree:

Decision trees are tree-like structures in where a class label is indicated by each leaf node, a test result is indicated by each branch, and a decision or test on a characteristic is indicated by each internal node. They are utilized for tasks involving classification and regression. We use it in classification [16].

4.3.4 Extra Trees:

Extra Trees (Extremely Randomized Trees) is a type of ensemble learning. method similar to Random Forest, but it further randomizes the feature selection and node splitting. It can improve diversity among trees and potentially reduce overfitting [16].

4.3.5 K-Nearest Neighbors (KNN):

KNN classifies data points based on their K nearest neighbors' majority class. It is non-parametric and suitable for simple classification tasks [16].

4.3.6 Lasso Regression:

Lasso is a regularization technique used in linear regression. Lasso adds the absolute values of coefficients to the loss function, encouraging sparsity [16].

4.3.7 Logistic Regression:

In spite of its name, logistic regression is a classification algorithm and not a regression algorithm. It models the probability of a binary outcome using the logistic function. It's simple, interpretable, and works well when the relationship between features and outcomes is roughly linear [16].

4.3.8 Multilayer Perceptron (MLP):

A multilayer perceptron (MLP) is a type of artificial neural network with multiple layers of nodes (neurons) that can learn complex relationships in data. It's often used for various tasks like classification and regression [16].

4.3.9 Naive Bayes:

Naive Bayes is a probabilistic classifier based on Bayes' theorem. Given the class label, it presumes that features are conditionally independent. It's widely used for text classification and other probabilistic tasks [16].

4.3.10 Random Forest:

Random Forest is an ensemble technique for improving accuracy and robustness by training multiple decision trees and combining their predictions. It reduces overfitting and generalizes well [16].

4.3.11 Ride Regression:

Lasso is a regularization technique used in linear regression. Ridge adds squared coefficients, which can help prevent overfitting [16].

4.3.12 Support Vector Machine (SVM):

SVM is a powerful algorithm for both classification and regression tasks. It works by locating a hyperplane that best separates classes in a high-dimensional space. It's effective in high-dimensional spaces and can handle non-linear data with the assistance of kernel functions [16].

4.3.13 XGBoost (Extreme Gradient Boosting):

XGBoost is a gradient boosting algorithm that builds an ensemble of weak learners (usually decision trees) sequentially, correcting the errors of previous models. It's known for its efficiency and high performance [16].

4.4 Performance Metric

In this section, we present the evaluation metrics used to evaluate the proposed system. Namely, we overall accuracy, recall, precision, F1 score.

- Overall Accuracy: measures how accurately a classification model predicts the appropriate class labels [16]. It is calculated as shown in equation 3 where TP, TN, FP, FP, and FN, respectively, stand for True Positives, True Negatives, False Positives, and False Negatives.

$$Overall\ accurracy = \left(\frac{TP+TN}{TP+TN+FP+FN}\right) \quad (3)$$

- Recall: focuses on minimizing false negatives [16]. It has an inverse relationship with FN. It is calculated as shown in equation 4.

$$Recall = \left(\frac{TP}{TP+FN}\right) \quad (4)$$

- Precision: focuses on minimizing false positives [16]. It has an inverse relationship with FP. It is calculated as shown in equation 5.

$$Precision = \left(\frac{TP}{TP+FP}\right) \quad (5)$$

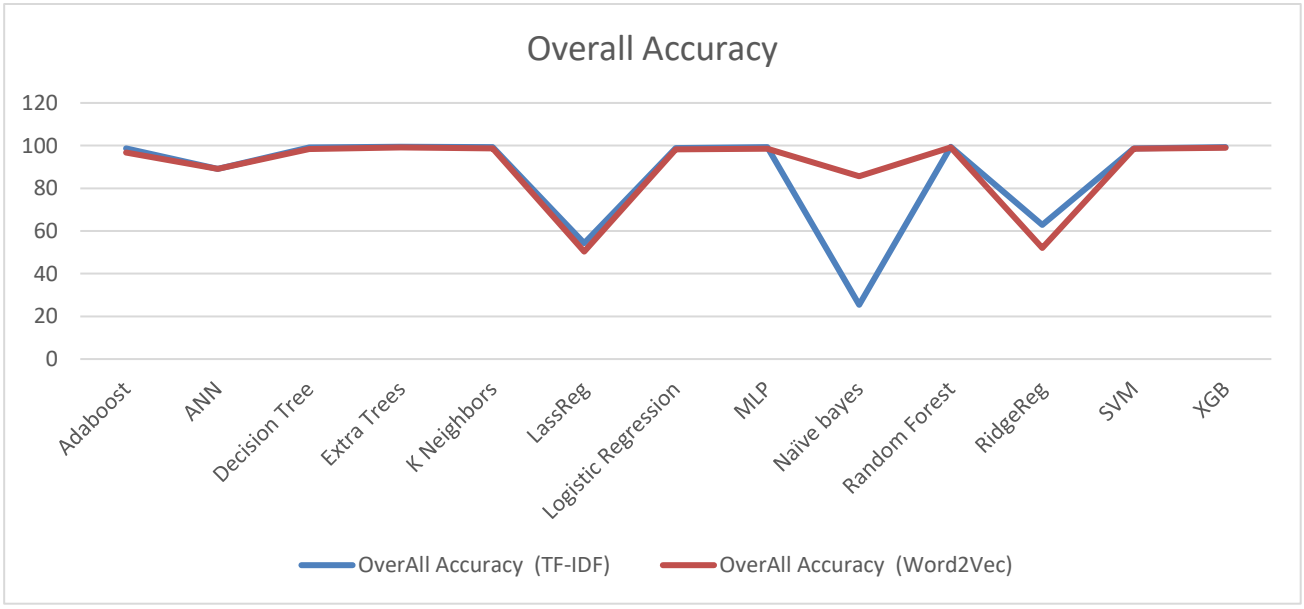
- F1 Score: provides a single score that accounts for both false positives and false negatives by taking the harmonic mean of precision and recall into consideration [16]. It is calculated as shown in equation 6.

$$F1\ score = 2 * \left(\frac{Precision*Recall}{Precision+Recall}\right) \quad (6)$$

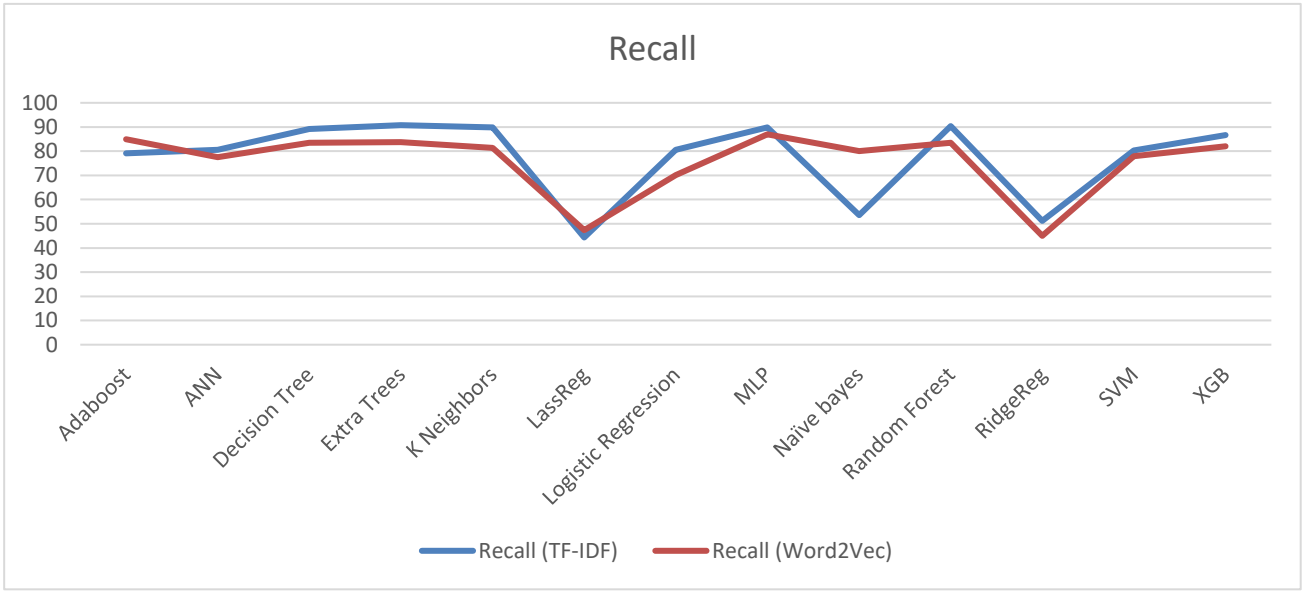
5. Experimental Results

In this experiment, we investigate using TF-IDF and Word2vec as two features extraction methods while applying the considered machine learning classifiers adaboost, ANN, decision trees, extra trees, K-nearest neighbors, lasso regression, logistic regression, MLP, naïve bayes, random forests, ride regression, support vector machines and XGB on the "Drebin" dataset. Figure 3 shows a comparison between the

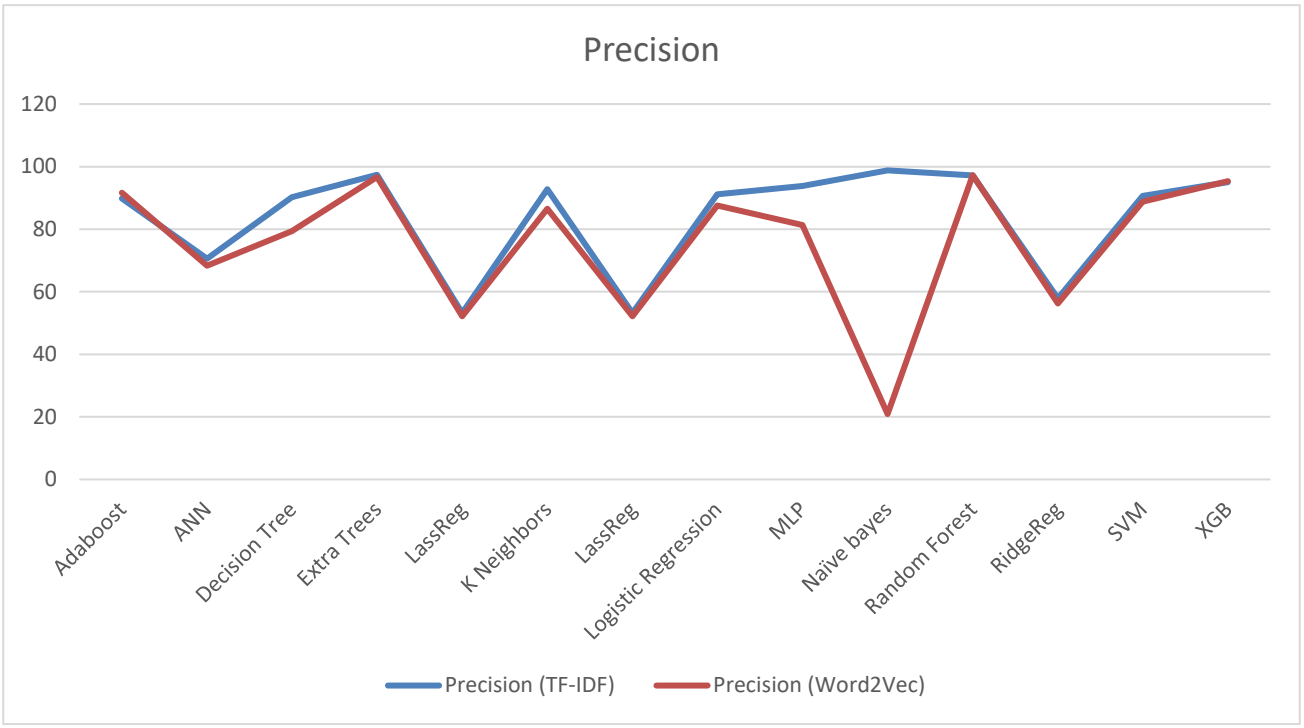
two feature extractors using different performance metrics overall accuracy, recall, precision, f1 Score and the execution time. Chart (a) shows that the overall accuracy of TF-IDF is superior than Word2vec with all classifiers except Naive Bayes. Chart (b) shows that the recall of TF-IDF is superior to Word2vec with all classifiers except Naive Bayes and adaboost. Chart (c) shows that the precision of TF-IDF is superior to Word2vec with all classifiers except XGB and random forest. Chart (d) shows that the F1score of TF-IDF is superior to Word2vec with all classifiers except adaboost. Chart (e) shows that the execution time of SVM with Word2vec is too long 5161 seconds. The results show that TF-IDF feature extractor performs better on the “Drebin" dataset.



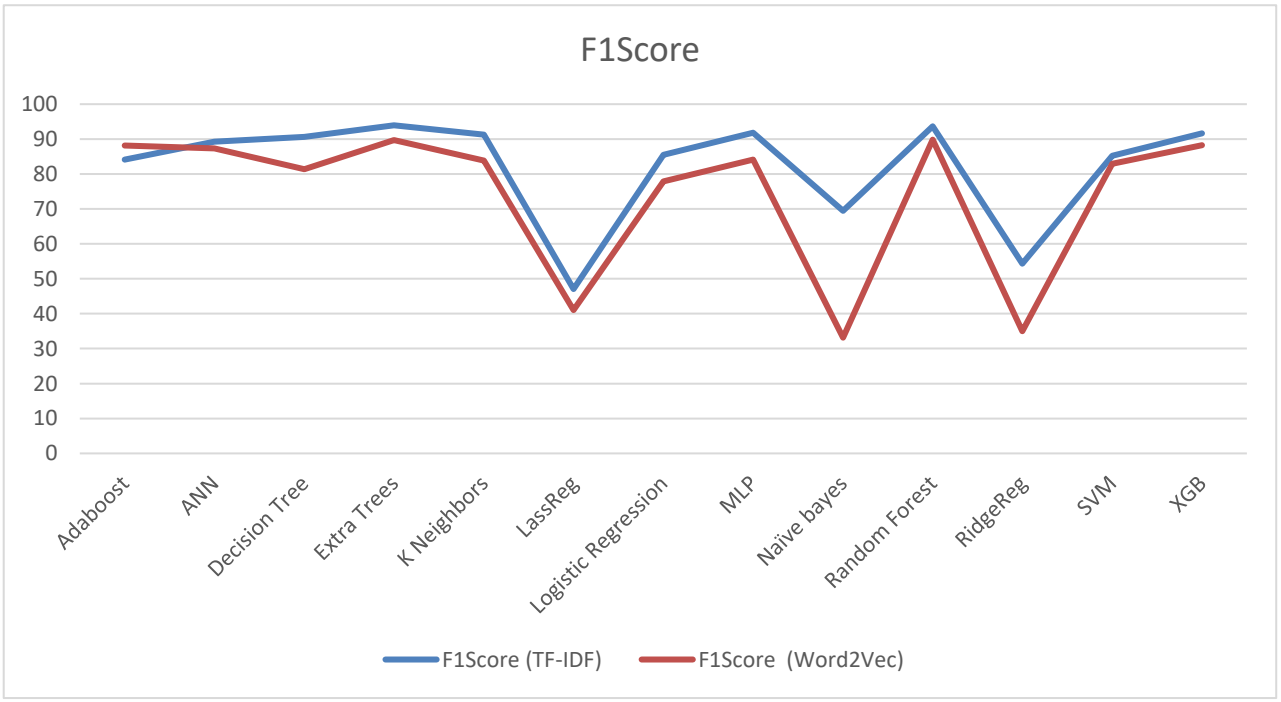
(a)



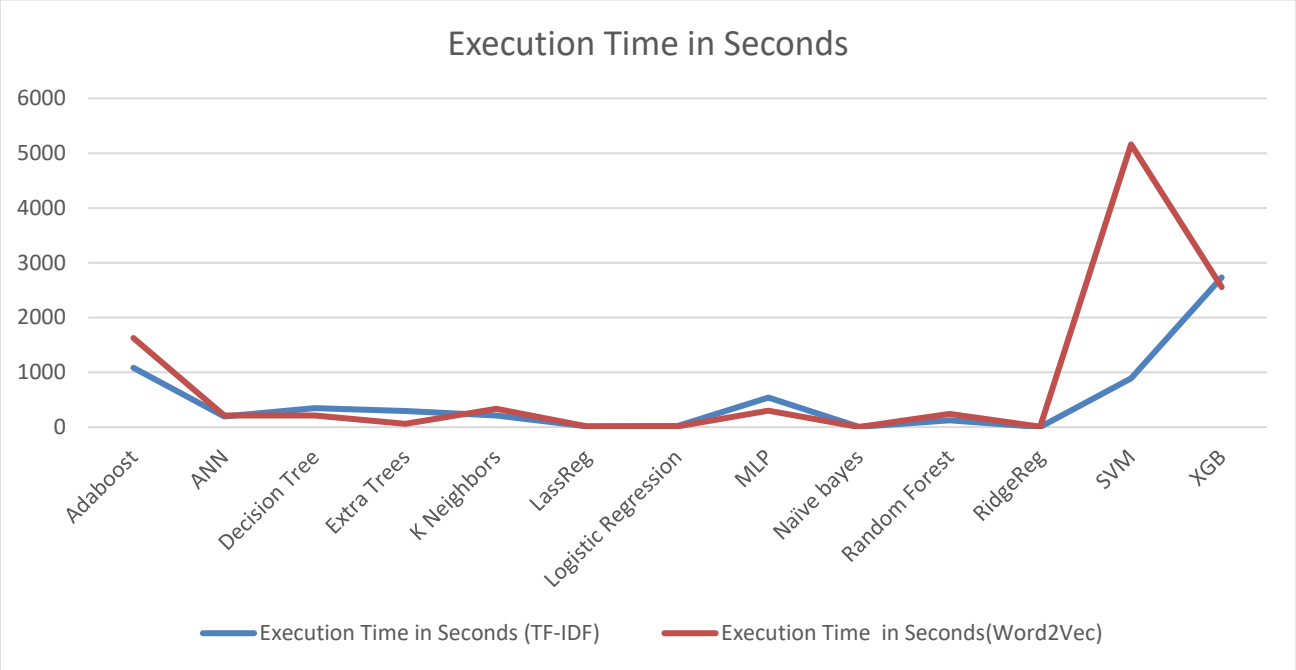
(b)


Precision
120
100
80
60
40
20
0
Adaboost
ANN
Decision Tree
Extra Trees
LassReg
K Neighbors
LassReg
Logistic Regression
MLP
Naïve bayes
Random Forest
RidgeReg
SVM
XGB
Precision (TF-IDF)
Precision (Word2Vec)

(c)


F1Score
100
90
80
70
60
50
40
30
20
10
0
Adaboost
ANN
Decision Tree
Extra Trees
K Neighbors
LassReg
Logistic Regression
MLP
Naïve bayes
Random Forest
RidgeReg
SVM
XGB
F1Score (TF-IDF)
F1Score (Word2Vec)

(d)



(e)

Figure 3: Comparison between TF-IDF and Word2Vec

6. Conclusion and Future Works

Malicious software or code targeted specifically at Android-powered devices is referred to as Android malware. These malware versions have the potential to steal data, inflict financial losses, and violate users' privacy. Static analysis has shown its robustness by preventing the installation of malware apps. Machine learning classifiers is a fast and an accurate way to detect malware from metadata (manifest file) of mobile application. We used different feature extractors and classifiers in "Drebin". We concluded that classifier accuracy increases with the increase of training data. Feature extractor contributes to an increase in accuracy. TF-IDF is the best feature extractor in "Drebin" dataset. Future work is developing an android antivirus based on "Drebin" dataset.

References

1. Pankaj Saraswat, "An inclusive analysis of Google's android operating system and its security", AIP Conference Proceedings 2427, 020097, 2023.
2. Wael F. Elsersy, Ali Feizollah and Nor Badrul Anuar, "The rise of obfuscated Android malware and impacts on detection methods", PeerJ Computer Science, 2022.
3. Choi S, Sun K, Eom H, "Android malware detection using library API call tracing and semantic-preserving signal processing techniques. Report".
4. Kang B, Yerima SY, Sezer S, McLaughin K, "N-gram opcode analysis for android malware detection", Int J Cyber Situational Awareness 1(1):1–24, 2016.
5. Wu S, Wang P, Li X, Zhang Y, "Effective detection of android malware based on the usage of data flow APIs and machine learning", Inf Softw Technol, 2016.

6. Rana MS, Gudla C, Sung AH, "Evaluating machine learning models for android malware detection: A comparison study", In: Proceedings of the 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, 2018.
7. Chen YM, Hsu CH, Chung KCK, "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, (2019).
8. Lou S, Cheng S, Huang J, Jiang F, "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, (2019).
9. Ma Z, Ge H, Liu Y, Zhao M, Ma J, "A combination method for android malware detection based on control flow graphs and machine learning algorithms", IEEE Access 7:21235–21245, (2019).
10. Shan P, Li Q, Zhang P, Gu Y, "Malware detection method based on control flow analysis", In: ICIT 2019: proceedings of the 7th international conference on information technology: IoT and Smart City, pp 158–164, (2019).
11. 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;15(3):209–18. https://doi.org/10.1007/ s11416-019-00332-z, (2019).
12. 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;7:64411–30, (2019).
13. Vij D, Balachandran V, Thomas T, Surendran R, "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, (2019).
14. 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, 102794, (2019).
15. 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.
16. Sara Mahmoud Shehata, Islam Hegazy, El-Sayed M. El-Horbaty, "Comparative Study for Andriod Mobile Static Analysis Algorithms", Journal of Theoretical and Applied Information Technology Vol 101 July 15, 2023, Issue 13.
17. Available online at https://developers.google.com/android/play-protect/phacategories, "Malware categories | Play Protect | Google for Developers, "Google Play Protect"", last accessed 12/7/2023.
18. Steve Anson, "Malware Analysis," in Applied Incident Response, Wiley, pp.277-309, doi: 10.1002/9781119560302.ch10, 2020.
19. Available online at https://www.aquasec.com/cloud-native-academy/cloud-attacks/malware-analysis/, "Malware Analysis: Static vs. Dynamic and 4 Critical Best Practices", last accessed 13/8/2023.
20. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.", Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS), 2014.
21. Available online at https://www.kaggle.com/datasets/fabriciojoc/fast-furious-malware-data-stream, "Fast & Furious: Malware Detection Data Stream", last accessed 11/8/2023.
22. Available online at https://en.wikipedia.org/wiki/Word2vec, "Word2vec", last accessed 1/8/2023.
23. Available online at https://en.wikipedia.org/wiki/Tf%E2%80%93idf, "tf–idf", last accessed 5/8/2023.

24. Available online at https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222, “Comparative Study on Classic Machine learning Algorithms”, Comparative Study on Classic Machine learning Algorithms, last accessed 12/8/2023.
25. Available online at https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/, “Beginners Guide to Artificial Neural Network”, last accessed 13/7/2023.