National College of Ireland

Advancing Android Malware Detection with Machine Learning Techniques

MSc Research Project
Cyber Security

Joseph Mathew
Student ID: 22151741

School of Computing
National College of Ireland

Supervisor: Dr. Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Joseph Mathew
Student ID:	22181741
Program:	MSc. Cyber Security **Year:** 2024
Module:	Practicum
Supervisor:	Dr. Imran Khan
Submission Due Date:	02/12/2024
Project Title:	Advancing Android Malware Detection with Machine Learning Techniques
Word Count:	7287 **Page Count** 22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Joseph Mathew
Date:	02/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Advancing Android Malware Detection with Machine Learning Techniques

Joseph Mathew
22181741

Abstract

Malware incidents are on the rise and are a great danger for information security. This project describes the application of a Random Forest technique for classifying applications into two groups: benign and malware. The model's real time predictive ability is further evaluated towards that end, and its ability in a cybersecurity context may prove valuable. The experiments show that Random Forest classifier can make effective malware detection, class distinction and accurate predictions. To this end, the real time predictive ability of the model is evaluated that may be helpful in a cybersecurity context. All the experiments reveal that Random Forest classifier has the possibility to be implemented for the detection of malware and can differentiate between the classes and make better predictions. More research may include improving the model to improve the hyperparameters, features selection, and managing class imbalance to enhance the performance of the model.

1 Introduction

Nowadays all the digital creations are receiving more attention than before since portability is growing at a geometric progression. This is mostly because of the daily growing mobile phone environment. There are more than two billion mobile phones in use globally today; this includes tablets, and feature phones.

Unfortunately, threats targeting Android based mobile devices have been on the rise as adoption of the devices rises rapidly. Malware that exploits vulnerabilities in the system is quickly and efficiently focusing on the Android operating system due to the widespread use of critical applications, for instance, banking applications.

To the detection of malware on mobile devices, several research proposed models. However, in other to effectively run and perform optimally more efforts must be made. Therefore, the current study applied machine learning techniques to detect the hostile attacks targeting Android.

However, with such measures already being taken to prevent contracting the infection on the growing base of Android users across the world, the spread of Android malware is quite worrisome. The Google Play store, which is the official store for Android apps, and several other unofficial online stores are the Market places through which most of Android apps are marketed. Android is such a famous and much utilized operating system because of the open-source nature of the platform that make app development easier. But being popular has a cost: problems of Android. Since the operating system is a free one, hackers have been able to use it as a host to their malware programs that are used in showing loopholes and other insecurity vices that infringe on the privacy of users [1].

However, this paper has shown that even with the enormous amount of investment that Google has made over the last few years to fight such applications, it is impossible to completely rule out these types of malicious intents. For checking mobile device behaviour and enabling successful and pertinent evaluations, it is necessary to name the tasks that may be performed and the kinds of details that can be obtained. The user must install the program and then enter his/her login details; the user must own an Android phone. Two or three functional and systemic requirements must be considered as well. Application monitoring begins after the user has started the application that is being checked. This involves the collection of data from the installed programs and operating systems in an Android based smartphone and displaying it to the user [3].

Among the collected data there is the number of starts of application per day which is also the launching time of the application.

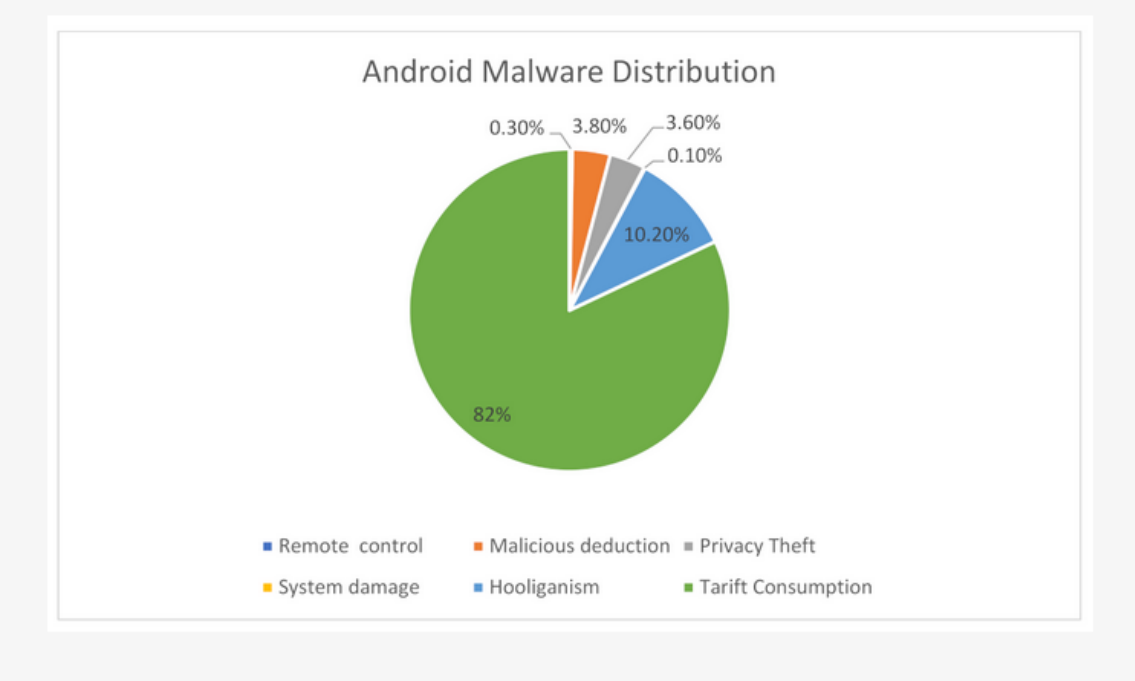

Fig.1 Percentage of android malware

As a result of customers' mobilization, the customers are able and willing to shift almost all their transactions that occur in their daily lives to the mobile environment and these applications. Thus, the application and development communities of the mobile devices may skyrocket. Mobile devices are regarded as highly individualistic instruments designed for the purpose of conducting routine tasks at the same time, same device maybe used to store very confidential data about an individual.

This research work aims to figure out the security threats relevant to mobile applications together with the best practices of security in this area. In the recent past there has been many internal and external security threats that have occurred, and mobile applications have had to face them. The risks concerning mobile application security involve a wide variety of threats that may potentially lead to the violation of the confidentiality, integrity, and available of the data owned by the user and the proper functioning of the application in the specific device. The main IT security risks are information disclosure, virus, and phishing.

Measures of enhancing the security assessment in the mobile environment can be extended by applying blockchain and artificial intelligence. Thus, it can be told that applying the artificial intelligence systems will increase the volume of the detected results and make the procedure more effective while analysing large data samples and patterns. Also furthermore, Machine learning (ML) and deep learning (DL) can be trained with the known vulnerabilities and these models can learn the pattern and abnormality of the Mobile Apps.

This assists in reducing the time that is employed to perform the analysis manually and avail developers with more time on the fix. The customers of mobile applications are gradually embracing security best practice on their gadgets. Some of these measures include restricting the permissions that apps have on the device, constantly updating various software on the device, and using strong passwords with supporting biometrics features such as the face and fingerprint identification. It showed that people are becoming even less willing to give permission to an app because they know more, and interfaces have improved. Even though data encryption is mostly set to be used automatically, some of the users do not connect to secure Wi-Fi networks, and therefore their data can be changed or stolen.
Although the use of phishing and social engineering is decreasing, there are still people who are being tricked by other more sophisticated methods.

However, most users prefer to download applications from authorized app stores; however, some users sideload apps, which increases the risk of getting malicious applications from untrustworthy sources. As we speak, it is essential to use lengthy, and unique passwords and password managers. However, there are still some opportunities for the improvement such as the technology, usability of security measures, and awareness.

In this research, we propose a machine learning algorithm to detect the malware in the dataset. This dataset is analysed using the machine learning algorithm random forest to predict the dataset which are malware infested. The dataset used for this research is "drebin-215-dataset-5560malware-9476-benign" and it have a more than 10,000 dataset which are malicious and non-malicious data [15].

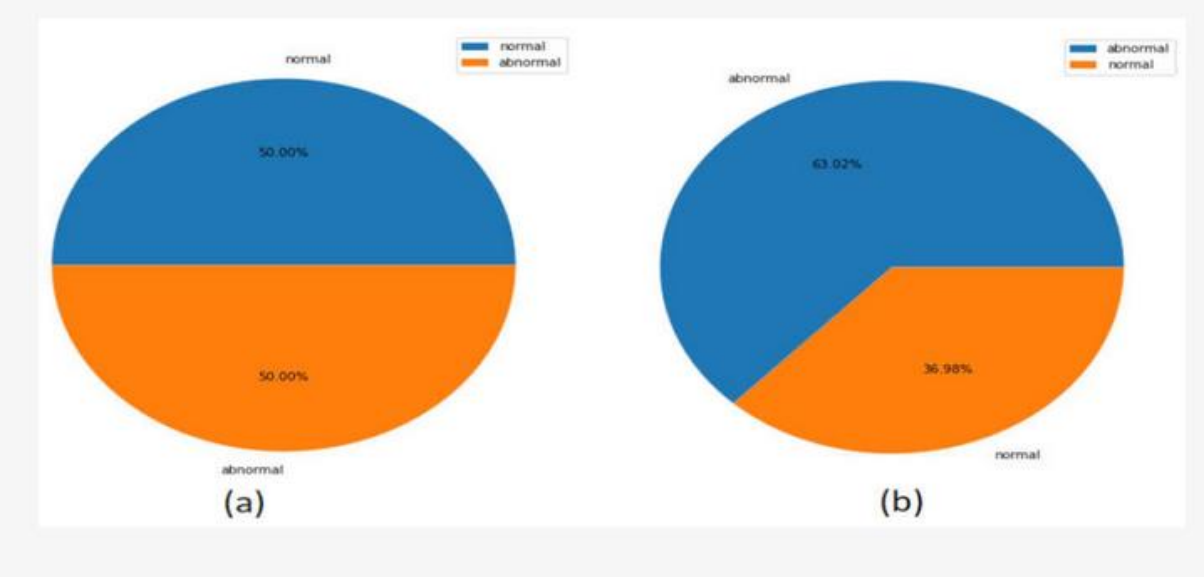

Fig.2 Percentage of class of dataset, (a) CICAndMal2017, (b) Drebin.

1.1 RANDOM FOREST ALGORITHM

Random Forest is a supervised machine learning algorithm which was discovered by Leo Breiman. The approach in this paper uses an ensemble of classification trees. A single result set is generated by the ensemble learning approach that creates multiple learners. Random Forest goes further than the Bagging technique. In

In the case of bagging, each of the classifiers is built up by using a bootstrap sample of the input data. At a node split in the conventional decision tree classifier a choice is made with all feature properties taken into consideration. However, in Random Forest, a random set of characteristics is used in the decision of the best parameter in each node of the tree. Besides that, random feature selection is useful in reducing the dependency that is often seen in the feature properties especially when there are several features in each feature vector for Random Forest models. Thus, this approach is relatively not sensitive to the data noise which is always present.

The Random Forest is one of the ensembles learning that based on the decision trees. During training, it builds a "forest" of decision trees, which trains a decision tree on a randomly chosen subset of the data and the 'features' (bagging, feature sampling). All of these tree's outputs are aggregated in the final prediction, normally through majority voting for the classification or averaging for the regression tasks.

Key Features:

- Robustness to Overfitting: RF avoids overfitting, and it does so by averaging multiple trees instead of using one single tree.
- Non-Parametric Nature: In contrast to other machine learning algorithms, RF makes no assumption about the underlying data distribution, and can therefore be used for learning complex, non-linear relationships.
- Manages High-Dimensional Data: The feature sampling strategy of RF allows it to perform well with lots of features.

For Selecting Random Forest algorithm over SVM or CNN are the following.

1. Data Type and Problem Scale:
    - RF works well on structured datasets (tabular data) and CNNs expect structures spatially, such as in images.
    - Finally, RF provides excellent performance with both classification and regression tasks, and SVMs would do well only for specific kernels with excessive cost for large datasets.
2. Ease of Use:
    - Again, RF is also less time consuming for hyperparameter tuning than SVMs or CNNs, making RF more friendly to practitioners.
    - Additionally, it features important scores, something that is hard to produce in the context of providing interpretability for an SVM or CNN.
3. Training Time and Complexity:
    - As a less deep architecture than CNNs, RF needs less computation than CNNs, which need a lot of resources in training.
    - RF is better scalable to the number of data points and their number of features than SVMs.
4. Data Requirements:

- This is because CNNs need large and labeled datasets to produce satisfactory results, of which we don't always have. However, RF fares much better with relatively smaller datasets.
- On the contrary, SVMs learn slowly about big data and imbalanced data, but RF does a much better job.

1.2 For Selecting Drebin dataset are for the following features:

Features of the Derbin-215 dataset are as follows: The Derbin-215 dataset has the following features:

Derbin-215 is the most appropriate of all the datasets for the analysis of the Android malware, since it contains unique samples. Below are the key specialties of the Derbin-215 dataset, explained in detail: Detailed Derbin–215 dataset key specialties presentation follows.

1) Comprehensive Composition

In total, there are 15,036 applications of which 5,560 are malware, and 9,476 are benign applications in Derbin-215 dataset. One gets these samples of 179 malware families and analyzed malware is quite diverse.

The researcher was able to find a vast number of behaviors and consequences of malware with this considerable number of samples.

2) Historical Data Collection

The data set was obtained from sample surveys carried out in August 2010 to October 2012.

This data is also useful in following the progression of Android malware, and so are effective in malware patterns and our stage of development identification in the search for better detection approaches.

3) Rich Feature Set

The nearly 215 features were derived from the applications on the Derbin-215.

APK attributes and permissions, and other textual features are used as these in classifying the malware from the normal samples.

This feature set is very helpful when training the intricate machine learning model as all notable features are included here.

4) Preprocessing Techniques

Some of the techniques used while data pre-processing includes APK to Image Conversion, Data Balancing, Normalization, Data Split, Label Encoding and Handling Missing Values.

The main purpose of these preprocessing steps is to prepare the data in the correct way, in order to allow machine learning models during training and testing, and hence boost the performance of such models.

5) Public Availability

Convenient use of Derbin-215 dataset provides an opportunity to evaluate Android malware and compare different detection methods. This accessibility allows research scholars all across the globe to use a dataset for experimentation and comparison of results with other models developed on the same dataset [17].6) The Proposed Model can be Utilized in Deep Learning Models. The Derbin-215 dataset has the following features:

Out of all the datasets, Derbin-215 is the most proper in the analysis of Android malware due to the presence of unique samples. Below are the key specialties of the Derbin-215 dataset, explained in detail: Here is a detailed description of the key specialties of the Derbin-215 dataset:

1) Comprehensive Composition

The Derbin-215 dataset has 15,036 applications in total of which 5,560 are malware and 9,476 are benign applications. These samples are obtained from 179 different malware families, and they are quite diverse in terms of malware that can be analyzed.

This considerable number of samples helped the researcher to find a vast number of behaviors and consequences of malware.

2) Historical Data Collection

The data set was obtained from sample surveys conducted in August 2010 to October 2012.

Such data is also helpful in seeing the evolution of Android malware and as such, can be highly effective in the identification of malware patterns and or stages of development in an attempt to come up with better ways of detection.

3) Rich Feature Set

The Derbin-215 has nearly 215 features that were derived from the applications.

These features include attributes of APKs and permissions and other textual features that are used in distinguishing between the malware and the normal samples.

This feature set is quite helpful in training the intricate machine learning models as all notable features are included here.

4) Preprocessing Techniques

Some of the operations that have been performed to the dataset in order to make it suitable for machine learning include Some of the techniques that are used during data pre-processing include APK to Image Conversion, Data Balancing, Normalization, Data Split, Label Encoding and Handling Missing Values.

These preprocessing steps help in preparing the data in the right format to feed the machine learning models for training and testing and hence enhance the performance of the models.

5) Public Availability

The Derbin-215 dataset is available for use, which makes it easier to analyze Android malware and compare different detection methods. This accessibility enables research scholars across the globe to use the dataset for the purpose of experimentation and comparison of results with other models developed on the same dataset [17].

6) Use of the Proposed Model in Deep Learning Models

Deep learning models have been trained and tested using the dataset for detection of Android malware. For example, a study conducted on the Derbin-215 dataset uses Convolutional Neural Network (CNN) model with 99% accuracy. It attained an accuracy of 92%, with high values of precision, recall and F1 score. That is to say, the dataset allows one to be able to use it to train complex machine learning algorithms for detecting malware [18].

7) Research and Development

The AMDDL model, a deep learning technique for Android malware detection, is developed using Derbin-215 dataset. Thus, the dataset is underlined as key in the improvement of the current state of the art in malware detection technologies. Therefore, the Derbin-215 dataset can be distinguished by the following advantages: the dataset is diverse and large, it contains historical data, features are quite rich, the data is prepared in great detail, the data set is public, and it has been used to craft high precision deep learning models. Due to the

foregoing specialties, it is a useful reference for researchers and implementers [19] in Android malware detection area.

Research Question: How to detect malware in a dataset using Random Forest algorithm.

2 Related Work

This section will present the current attempts to provide solutions for enhancing the security analysis process in the mobile environment. The focus will be on machine learning-based approaches with existing machine learning-based approaches as the second source of solutions.

- The research done in this paper is a novel approach to the classification of Android malware which considers both static and dynamic characteristics of the malware applications. It has Naïve Bayes, Decision trees, Random forests, among other machine learning algorithms. Random forest is the most exact model that has a classification rate of 84% for the identification of the family of malware from mobile applications.

- The authors' approach for detecting mobile malware calls for an architecture based on machine learning. The framework uses static analysis of Android apps to extract features. The SVM and RELIEF algorithms were the two feature selection techniques that were applied.
- Published in 2013 by Borza et al., the textbook PUMA: This book was written by Borza et al and published under the title PUMA in the year 2013.Permissiveness on the identification of Malware in Android Operating System. This course presents a new way of recognizing evil; This paper looks to present the problem of employing machine learning methods in the identification and assessment of the Android application's intricacy. instead, it focuses on the other permissions granted to the programs.
- The paper on malware detection that Zarni Aung et al. (2013) developed is titled Permission-Based Android Malware Detection.
  Thus, for enhancing the security and privacy of the mobile users in this research, a framework for the Android malware detection system has been proposed.

- The study describes a method for protecting mobile devices from infection. This study proposes a novel malware detection algorithm for mobile device security. Support Vector Machine (SVM), a machine learning algorithm, is shown in a study. Paper provides an inventive method for naming malware in Android applications. Models for Android OS malware detection is presented in the article. The work describes a paradigm for recognizing and blocking dangerous behaviour in Android applications. In this study, a modelling framework and method are suggested. For modern mobile devices, the study proposes a high-accuracy

malware detection method. This research presents a framework for analysing user behaviour based on mobile phone activities.

3 Research Methodology

Nowadays, use of mobile technology is inevitable part of our everyday life. There are many applications that are used most of the times, in business or in leisure. The security aspect of mobile app and the personal information being shared are not something most people focus on.

The user's behaviour and other contextual data can be aggregated to create behavioural models to differentiate the benign from the malignant behaviour of the user. Then, a remote server-based machine learning system is fed the properties of this collected dataset. It is even possible to make the IDS for the future work in real time on a mobile device.

There are two methods for finding malware on computers: There are two categories of such analysis: static analysis and dynamic analysis. Dynamic analysis in this case executes the file like a computer or even through a sandbox tool to extensively examine the malware's behaviour, whereas static analysis monitors the functionality of an application file without running it. In current reality, around machine learning, for malware for Android devices with common evasion techniques, each present work concentrates on dynamic analysis. Static analysis does not do any good in this case because malware might leave 'footprints' coming under attack with techniques like packed encryption, but static analysis can consider techniques like packed encryption and thereby reduce the impact of malware.

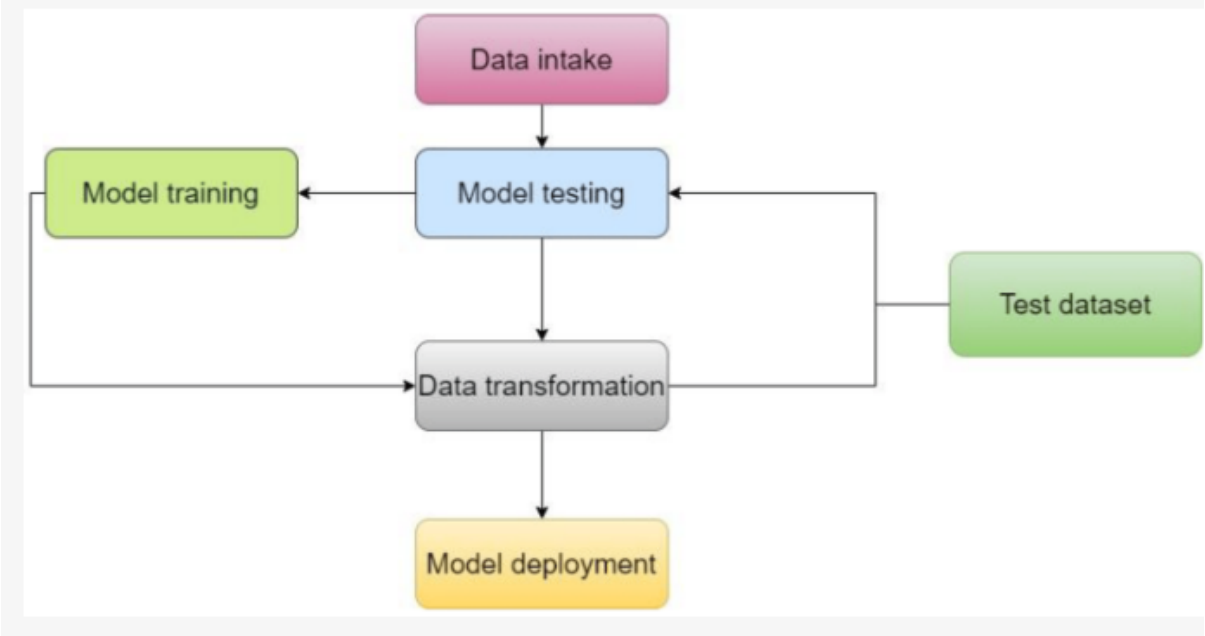

Fig.3 Workflow process

Another type of malware that has been on the rise these days is what is commonly referred to as botnets. What was earlier used to affect PCs is now affecting smartphones, and other mobile devices. The botnets that affect the device networks can act independently and become a threat to the mobile environment. Botnet attacks are now becoming common on

Android operating system devices and most of the times, the attack is done silently. Sensitive information can be compromised through DDoS attacks such as HTTP Flood, Ping Flood and many more irrespective of the mobile application. Even though the ML detection was designed to prevent DDoS attacks, up to the present, it has not been implemented in the mobile sector. The first of the stages of use of the Random Forest algorithm is the so-called 'loading and data preparation' stage.

The process that was followers in the research are:

3.0.1 Loading the Dataset: The data is in the form of CSV file and thus to read the data the function used is the read_csv () from the pandas which helps in creating a data frame.

Preprocessing the Data: This is often called the 'label', and it is assumed that the label is in the next column. For this work, the predictors are found in a variable referred to as X while the results are found in variable referred to as y.

This is done to prepare it for training on the features and labels and for using only the features and labels to train in the course of the evaluation.

```
In [10]: file_path = '5561.csv'
```

Fig.4 File path

3.0.2 Splitting the Dataset

For this purpose, the data set is employed in partitioning of the training and testing set with the view of assessing the suitability of the model.

Train-Test Split: To split data into training and evaluating the function used is Train_test_split and this can be found in the scikit-learn toolbox. In general, the division of the data is 80/20 for training and testing of the model [15].

This is done in a fashion that part of the data is used in training the model, while the other part is used to assess the performance of the model on fresh data, data it has not come across before.

```
X = data.drop(columns=['class'])

y = data['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Fig.5 Splitting dataset.

The Creation of the Random Forest Model; The Alteration of the Same

It describes what the Random Forest model is and as to how such a model is built, the training dataset is used.

3.0.3Model Initialization: It has been used with n_estimators set at 100 and random state so as to give out results that can be reused.

It is used to increase the compatibility of the output with the further runs of the program via the random state. To make the results of the model like the ones obtained in various runs, a random state of the system is assigned.

3.0.4 Training the Model: To get the fitted model, the fit method is called which has two parameters the training features and training labels.
Training is carried out with the construction of several decision trees and each of the tree is constructed from different training samples and attributes.
3.0.5 Making Predictions: It is now time for the testing set to be used in a manner of producing the predicted outputs with the help of the trained model.

```
: rf_model = RandomForestClassifier(random_state=42)
```

Fig.6 Training Rf model.
3.0.6 Predictions on Testing Set: The predictions are made on testing set (X_test) and the same is done by using the predict function of the model.

The model runs with the help of a voting system that is used for the classification of an object under test into a particular class.
3.0.7 Model Evaluation: For the assessment of the model numerus evaluation metrics are used.
3.0.8 Accuracy Score: To evaluate the performance of the model the function accuracy_score is used to find the proportion of correct predictions performed by the model.

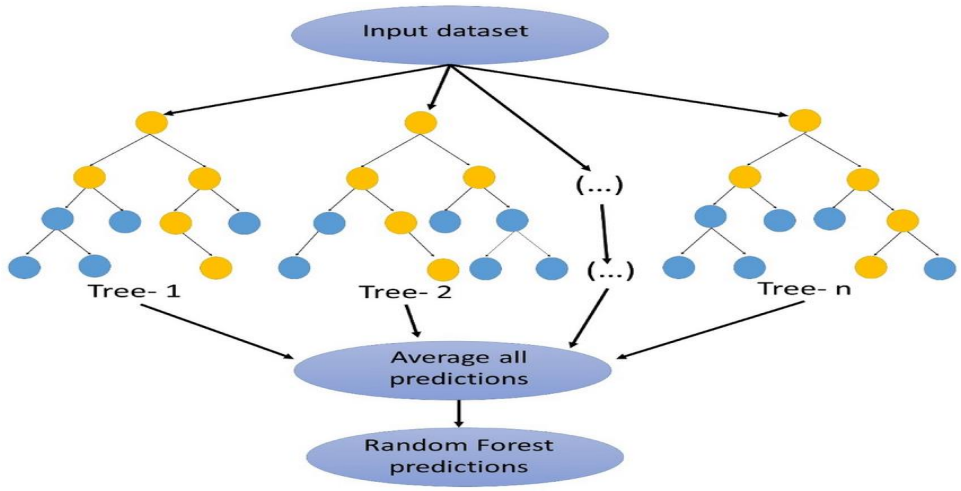

Fig.7 Diagram of overview of Random Forest prediction.

3.1 Materials and Methods.

The choice of the machine learning algorithm for the identification of malware on a mobile device depends on the type of malware, features being used, and the characteristics of the given platform. However, the following techniques are commonly used for this purpose: Nevertheless, the following are some of the techniques that are used in this regard:

- Random Forest: This adaptable algorithm is resistant to overfitting and capable of handling high-dimensional data. For classification jobs like malware detection, it performs admirably.
- Support Vector Machines (SVM): SVM can oversee high-dimensional data and is useful for binary classification jobs. When the boundaries between classes are well defined, it functions effectively.

- Convolutional neural networks, or CNNs, and recurrent neural networks, or RNNs, are two types of deep learning models that have proven promising results in a variety of ability to automatically extract features from raw data, which is useful for showing intricate virus patterns.

- Ensemble Techniques: The models are combined using assembling like AdaBoost and Gradient Boosting to increase the model accuracy in the given problem. It performs good when one model has an advantage over the other in some aspect.

- Using Long Short-Term Memory (LSTM) for Deep Learning: LSTM is suitable for sequential data analysis and can be used to set up whether a mobile device's behaviour is suggestive of malware presence.

4 Design Specification

In this the machine learning build with the help of Anaconda. And the extracting and splitting the dataset using jupyter notebook with python version. A Python library called NumPy is used for numerical operations. Moreover, arrays can be created and changed with it; this is especially useful for label storage. A matrix of TF-IDF features can be created from a collection of text documents using TfidfVectorizer. A static numerical measure called Term Frequency-Inverse Document Frequency is intended to stand for the way data is to be gathered. It also aids in transforming permissions into a format that is proper for machine learning.

4.1 Benefits of NumPy and TfidfVectorizer Libraries.

Advantages of Using NumPy

1. Performance and Efficiency: It is a package that includes powerful, optimized and efficient solver for numerical operations. It has support for large multi-dimensional arrays and matrices, and it has a collection of mathematical functions which are used on these arrays. For this reason, NumPy is quite useful when it comes to calculations and operations on number, vectors, and matrices and linear algebra.
2. Integration with Other Libraries: NumPy is intricately linked with other scientific computing libraries in python including the SciPy, the Pandas and the scikit-learn. It is also extremely helpful in the sense that it enables one to manipulate data, features and models at once particularly in the case of data pre-processing, feature selection and model building.
For instance, it is very convenient to shift data from one structure to the other; say from a NumPy array to a Pandas data-frame or from a data-frame to a NumPy array with the aim of processing data.
3. Memory Efficiency: It is because in Python when it comes to lists, the memory that is needed is more than when it comes to NumPy arrays. They are more economical and, in most cases, more efficient in the management of large data sets. This is immensely helpful especially in high dimensional data set because it is assuring that the operations are not only done in the least time possible but also with the least memory consumption.

Advantages of Using TfidfVectorizer

1. Effective Feature Extraction: TfidfVectorizer is one of the best techniques that can be used in transforming the given text data into numerical feature. This class has the combined functionality of the Count Vectorizer and TfidfTransformer and therefore it is easier to use when transforming textual data into vectors that can be used by ML algorithms.
This is useful for tasks like text classification, sentiment analysis and topic modelling.

2. Quantifying Word Importance: TF-IDF is a quantitative way of measuring the importance of a word in a document or a corpus of documents; it is an acronym for Term Frequency-Inverse Document Frequency. TfidfVectorizer is beneficial in word frequency, and this is very vital in information retrieval, document clustering, and text categorization [13].

3. Filtering Out Stop-Words: To avoid such words, there exist TfidfVectorizer by which stop-words are removed – the words which carry negligible importance (like 'the,' 'is,' 'in'). This is useful in the sense that it helps to reduce the size of the feature space and thus it improves the efficiency of the model.

4. The only other factor is Simplicity and Ease of Use: Tfidf can be implemented with scikit-learns Tfidf Vectorizer and it is quite easy to use. This conversion to TF-IDF features is rather straightforward and can be accomplished in a matter of three or four lines of code as such, anyone can perform it, including those with a little or no background in natural language processing, NLP [14].
Likewise, scikit-learn provides better documentation and comprehensive materials for TfidfVectorizer and therefore easier to use.

5. Consistency in Preprocessing: That is why saving the fitted TfidfVectorizer for the future use is useful – it will help to have the same way of data preprocessing for the text data in the future or in other applications. This is especially useful in real life, when one wants to apply some machine learning algorithms to some new data, because it guarantees that the data will be transformed in the same way every time [13].

Hence, in the case of the specific machine learning code, the use of libraries such as NumPy and TfidfVectorizer have been seen to have several merits. It is because of the following reasons that make NumPy especially useful for numerical operations: It is important for NLP tasks because it helps in-feature extraction from text, word weighting, stop words removal and text normalization. It is possible to use these libraries to develop machine learning models that are exact, and which do not take a lot of time to train [12].

5 Implementation

The code involves data input, data cleaning, model selection which is the Random Forest classifier, prediction and evaluation of the model. Below are the steps that will be followed to effect this: The following procedures will be put in place to implement this plan:

to load the dataset

```
In [3]: import pandas as pd
In [4]: from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
In [5]: from sklearn.preprocessing import StandardScaler, LabelEncoder
In [6]: from sklearn.ensemble import RandomForestClassifier
In [7]: from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
In [8]: import matplotlib.pyplot as plt
In [9]: import numpy as np
```

Fig.8 Loading libraries.

pandas: For the instances of data manipulation and loading. sklearn. model selection: train_test_split: For use in splitting the data into training and test sets. GridSearchCV: For model selection with hyperparameters tuning and cross validation. cross_val_score: While using cross validation for model assessment. sklearn. Ensemble: RandomForestClassifier: As for the Random Forest model that is used for classification performed. sklearn. Preprocessing: Label Encoder: Maps categorical target values into '0' and '1'. StandardScaler: transforms features in a standard way by making the mean equal to 0 and variance equal to 1. sklearn. Metrics: classification report: Summary of classification metrics which includes – precision, recall and F1score. confusion matrix: Shows actual vs. forecasted categories. roc_auc_score: Calculates the ROC AUC score. matplotlib. pyplot: Hi that means you get it for: Plotting feature importance. NumPy: Ablates mathematical computations through numbers and array computation aid.

```
In [10]: file_path = '5561.csv'
```

Fig.4 file path.

There is a dataset that we load into a panda Data Frame. Instead, use the real location of your dataset instead of file path.

- Split the Dataset into Features and Target Variable: X is all feature columns minus class (dropped).

x holds the features, and y holds the target column class which stands for whether the file is malware or safe.

- Split Data into Training and Test Sets: Splits the data into 70% training, and 30% testing.

stratify=y means that the two sets have same distribution of the target classes.

```
X = data.drop(columns=['class'])
```

```
y = data['class']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Feature Scaling: StandardScaler subtracts the mean and scales to unit variance of feature values to improve model performance.
On the training set, fit transform is used while on the test set transform is used to avoid data leakage.

- Define the Initial Random Forest Model: Random State is used to fix a random state so that the Random Forest Model can be initialized with a fixed random state to make the results reproducible.

Set Up Parameter Grid for Hyperparameter Tuning:
Specifies values for various Random Forest parameters:
n_estimators: Quantity of trees in the forest.
max_depth: Maximum depth of each tree.
min_samples_split: Number of samples required in each node to split that node.
min_samples_leaf: Number of smallest samples needed in each leaf node.

- Perform Grid Search with Cross-Validation: GridSearchCV will do exhaustive search over the parameter grid and see what combination works best with cross validation (cv=5).

roc_auc = scoring='roc_auc' which means maximize AUC score, which means that the goal of scoring is to separate classes.
For quicker execution, parallel processing is performed if n_jobs=-1.

- Best Parameters: GridSearchCV's outputs best parameter combination found.

- rain the Optimized Random Forest Model: It initializes and trains Random Forest model using the best values of parameters among the grid searched.

- Cross-Validation Scores: It measures cross validation scores with the ROC AUC metric and returns averages the scores across the folds.

Predictions and Evaluate:

1) Predictions:

y_pred: Malware vs. safe binary predictions.

y_pred_proba: The probabilities for each prediction being malware.

2) Evaluation:

classification report: Including precision, recall and F1 score of summaries.

confusion matrix: True vs. predicted class comparison.

roc_auc_score: Score of assessing the model's probability of classifying an observation correctly based on its probability of one class or another.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      2843
           1       0.99      0.97      0.98      1668

    accuracy                           0.99      4511
   macro avg       0.99      0.98      0.99      4511
weighted avg       0.99      0.99      0.99      4511

Confusion Matrix:
 [[2828   15]
 [  47 1621]]
ROC AUC Score: 0.9984784244359701
```

Fig.9 Output of precision, recall and F1 score of Accuracy, Macro Average and weight average. The Confusion Matrix and ROC AUC Score.

- Feature Extraction: Each feature's importance score is listed in feature importances.

The 14 most important features are extracted by sorted_idx.

- Plotting Feature Importances: Plots a bar chart of top 14 most important features.

plt. barh makes a horizontal bar plot: each feature's importance score.

The features are labelled by plt. yticks.

Additional Considerations

- Hyperparameter Tuning

Random forest also has its default hyperparameters and may not be the best for all the datasets. Some of the parameters that can be adjusted include n_estimators, max_features, min_samples_leaf and so forth; the use of grid search or random search is recommended for this purpose with a view of improving the performance of the model.

```
In [21]: param_grid = {
             'n_estimators': [100, 200, 300],
             'max_depth': [None, 10, 20, 30],
             'min_samples_split': [2, 5, 10],
             'min_samples_leaf': [1, 2, 4]
         }
```

Fig.10 Best parameters

```
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, scoring='roc_auc')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_depth': [None, 10, 20, 30],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [100, 200, 300]},
             scoring='roc_auc')
```

```
estimator: RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
best_params = grid_search.best_params_
```

```
best_score = grid_search.best_score_
```

```
print("Best Parameters:", best_params)
print("Best Cross-Validation ROC AUC Score:", best_score)
```

```
Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Cross-Validation ROC AUC Score: 0.9979906499707261
```

```
X = data.drop(columns=['class'])
```

```
y = data['class']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

Fig.11 Best parameters output.

- Feature Importance

Random Forests can provide feature importance which in its turn can help to define which of features is more critical for the given prediction. This can be done with the feature importances_ attribute of the trained model for instance.

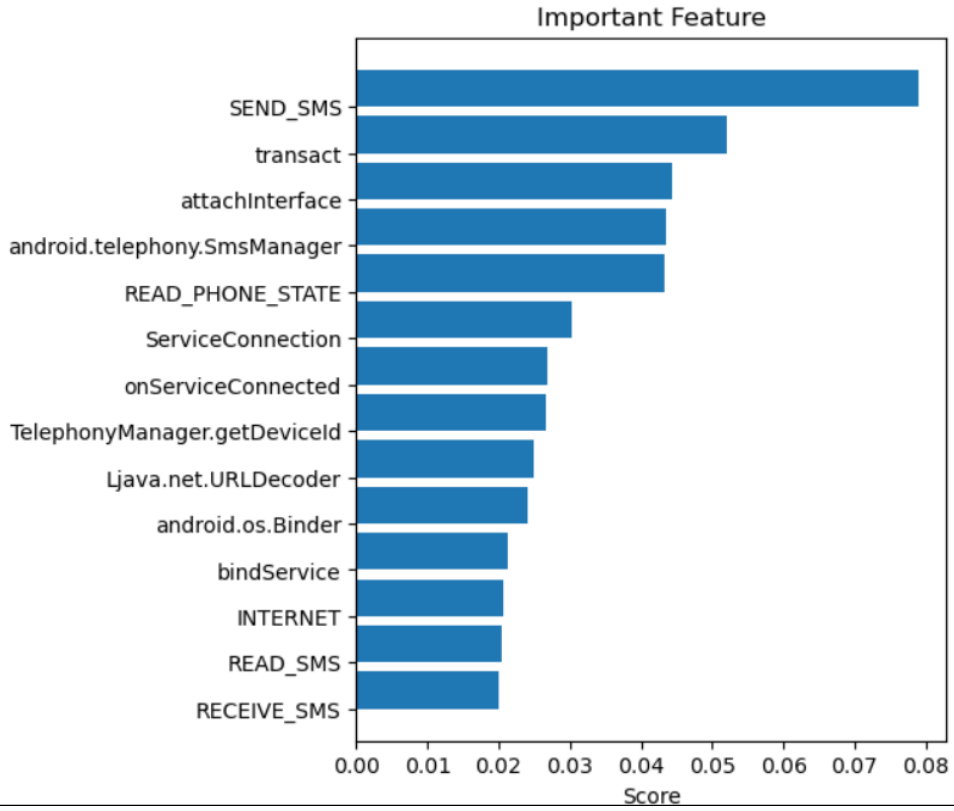

Fig.12 Important Features.

- Managing Categorical Data

Random Forests have few issues with high cardinality categorical features. Features such as these are better transformed prior to apply one hot encoding such as through targeting encoding or embeddings rather than one hot encoding.

- Dealing with Outliers

Outliers and normalization are the first step that can be applied to enhance the accuracy of the Random Forest model. The effectiveness of these steps is, however, largely determined by the quality of the initial data.

To implement Random Forest model in Python using Scikit-learn the steps include loading data, feature selection, data splitting, building the model, prediction and evaluating the model. After doing the above steps and the hyperparameter tuning tips and features selection tips, one is in a helpful position to develop a good and effective Random Forest model.

6 Evaluation

Let's compare the two of them: non malware (benign) and malware programs. As such, these samples must be stored by their feature vectors, where a feature is a behavioural property of the program which may evolve in a malware. There is something about the dataset. For example, structure of code, access to the file system by an application, network usage, types of API calls that an application makes, permissions required by the application can all be its features. To train the Random Forest model for the classification of Malware

and Benign programs there are features which needs to be extracted. As the number of malicious samples can greatly exceed the number of benign samples, the dataset must be pre-processed, which will be focusing on completing missing values, class balancing and features scaling. Split the set into two, such that one is the test, and one is the train. The Random Forest model should be trained by using the training data set made of the samples that they are classified as malicious or benign. With the

Thus, the app features obtained from the procedure will assist the app model to classify the apps. It is by that which we need to examine the trained model using the testing set. With F1 score, recall, ROC AUC score, confusion matrix, accuracy and precision used as metrics, to determine the degree of how effective the malware detection is provided with a low false alarm rate. Accordingly, when the model has achieved the intended level of performance, it is good to use the model on the mobile device. When I say implement the model, it could mean take the system and integrate the model into mobile security application system that could scan the installed programs for malwares. For the newly launched or new applications that are being deployed or installed into a mobile device, they should document them well in their properties and introduce the data into the trained real time Random Forest model. Based on these learned patterns the model then estimates if the program is likely to be a malware or not. Based on the given model, it will inform you if you should prevent that application from being installed, or to inform the user about the possible risk or put the suspected applications in quarantine to be reviewed later. Since the model may become useless in the future, and counter the new kinds of malware threats, it is suggested to periodically check the model's performance to give updates with new features, and datasets.

In this paper, Random Forest is used, and the algorithm performance of this study is evaluated. In this paper, applications of the method of deciphering the type of Android applications based on their malware status using machine learning techniques, are demonstrated. On the experiment we observed that multiclass classifiers outperformed other methods in classification. Preventive measures of removing malware by ML models. Though as explained before, a lot remains to be done with respect to the feature extraction step. When the activity is originated from intelligent Android mobile devices, the Random Forest algorithm scores the best in categorizing the activity as malicious out of these. From now on, this system is supposed to be used for securing Android mobile of an individual. Consequently, we need to have much larger and a much more varied dataset since this form of dataset could suddenly not be available for us.

7 Conclusion and Future Work

Random Forest is one of the techniques that can be used to build a model that is to be used in the classification of malware. The evaluation of the proposed model can be done using performance metrics such as the confusion matrix, accuracy, and classification report. These metrics provide a reasonable measure of the recall, the accuracy, the precision and the F1 score of the model for each class. However, the model has the problem of class imbalance, it may not be very efficient for all classes of the given data. As we can see from the classification report and the Confusion Matrix the model that has been selected has certain issues with the classification of certain classes. It also must be noted that the quality of the data that is used in the model is directly proportional to the quality of the model’s output and hence the importance of data quality. To ensure that the predictions that need to be made can

be as correct as possible it is important to note that the data set used in this case has been cleaned and pre-processed.
Overall, it was quite possible for the Random Forrest algorithm to effectively distinguish between the “malware” cases from normal ones as seen from the given set.

As for the future work, some enhancements have been made to the Random Forest model in a few areas. Some of them are feature engineering, dealing with imbalanced data, dealing with outliers, dealing with hyper parameters, feature selection, ensemble methods, model stacking, dealing with categorical variables, advanced data cleaning and pre-processing, and interactive data visualization and report.

To improve the performance of the model there is need to perform advanced data cleaning and preprocessing where new feature are created from the existing features. To tackle the problem of imbalanced data, some of the techniques used include by using Imbalanced-Learn to perform techniques such as oversampling the minority class, or under sampling the majority class. Outlier analysis to test its impact on the model and new age techniques to detect and deal with them are part of the Outlier detection and management.

Hyperparameter tuning is about doing thorough hyperparameter tuning using either a grid search or random search. Explanations for the choice of the feature importance analysis is that it aids in feature and model selection and tuning. Ensemble methods and model stacking take the best from several models, while the techniques for working with high cardinality categorical variables can enhance the model’s accuracy and speed.

Ultimately, the following areas are the future work of the Random Forest model: the improvement of the model, the increasing of the model’s stability, and the application of the model in real-world problems.

References

[1]. Samia El Haddouti; Anass Betouil; Habiba Chaoui "Enhancing Mobile Security: A Systematic Review of AI and Blockchain Integration Strategies for Effective Analysis" https://ieeexplore.ieee.org/document/10391475/references#references
[2]. Seema Vanjire; M. Lakshmi Behavior-Based Malware Detection System Approach for Mobile Security Using Machine Learning https://ieeexplore.ieee.org/document/9671009?arnumber=9671009
[3]. H. Alkahtani and T. H. H. Aldhyani, "Artificial intelligence algorithms for malware detection in android-operated mobile devices", https://www.mdpi.com/1424-8220/22/6/2268
[4]. Paweł Weichbroth; Łukasz Łysik; "Mobile Security: Threats and Best Practices" https://www.hindawi.com/journals/misy/2020/8828078/
[5]. Nayeem Islam, Saumitra Das and Yin Chen, "On-Device Mobile Phone Security Exploits Machine Learning", https://ieeexplore.ieee.org/document/7891134
[6]. A. A. T. Pektaş, Ensemble machine learning approach for android malware classification using hybrid features, vol. 578, March 2018. https://www.researchgate.net/publication/316799719_Ensemble_Machine_Learning_Approach_for_Android_Malware_Classification_Using_Hybrid_Features

[7]. G. F. I. N. V. I. Geneiatakis and D. Baldini, Towards a mobile malware detection framework with the support of machine learning, vol. 821, March 2018. https://link.springer.com/content/pdf/10.1007/978-3-319-95189-8_11.pdf
[8]. Khurram Majeed1, Dr Yanguo Jing2, Dr Dusica Novakovic3, and Prof Karim Ouazzane "Behaviour Based Anomaly Detection for Smartphones Using Machine Learning Algorithm" http://iieng.org/images/proceedings_pdf/2550E1014073.pdf
[9]. Is there a performance difference between NumPy and Pandas? https://stackoverflow.com/questions/21567842/is-there-a-performance-difference-between-numpy-and-pandas#:~:text=I%20was%20wondering%2C%20is%20there%20a%20difference%20in%20computational%20ability%20when%20using%20Numpy%20vs%20Pandas
[10]. How can Python be used for data preprocessing in Machine Learning? https://www.linkedin.com/advice/1/how-can-python-used-data-preprocessing-machine-ywacf#:~:text=some%20of%20the%20most%20common%20Python%20libraries%20for%20data%20preprocessing%20in%20Machine%20Learning%2C%20such%20as%20pandas%2C%20numpy%2C%20sklearn%2C%20and%20scipy
[11].TF-IDF Explained and Easy Examples in Python to Get Started https://medium.com/@neri.vvo/tf-idf-explained-and-easy-examples-in-python-to-get-started-a32d46f857a9#:~:text=Tf-idf%20allows%20text%20to%20be%20turned%20into%20numerical%20vectorizes%2C%20which%20is%20crucial%20for%20many%20machine%20learning%20algorithms%20that%20only%20work%20with%20numerical%20input
[12]. TF-IDF Guide: Using scikit-learn for TF-IDF implementation https://www.capitalone.com/tech/machine-learning/scikit-tfidf-implementation/
[13]. How to store a TfidfVectorizer for future use in scikit-learn? https://www.geeksforgeeks.org/how-to-store-a-tfidfvectorizer-for-future-use-in-scikit-learn/
[14]. Random Forest in Python https://towardsdatascience.com/random-forest-in-python-24d0893d51c0
[15]. Random Forest in Python (and coding it with Scikit-learn) https://data36.com/random-forest-in-python/#:~:text=which%20we%E2%80%99ll%20make%20it%20possible%20to%20randomly%20separate%20our%20dataset%20into%20train%20and%20test%20data
[16].A Practical Guide to Implementing a Random Forest Classifier in Python https://towardsdatascience.com/a-practical-guide-to-implementing-a-random-forest-classifier-in-python-979988d8a263#:~:text=We%20can%20instantiate%20it%20and%20train%20it%20in%20just%20two%20lines
[17].Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls https://www.researchgate.net/publication/336955434_Extensible_Android_Malware_Detection_and_Family_Classification_Using_Network-Flows_and_API-Calls
[18]. CYBERSECURITY DATASETS http://ahlashkari.com/Datasets.asp
[19].AMDDLmodel: Android smartphones malware detection using deep learning model https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0296722#:~:text=The%20used%20dataset%20name%20Drebin%20dataset%20%5B28%5D%20consists%20of%20almost%20215%20feature%20attributes%20extracted%20from%2015%2C036%20applications%20with%205%2C560%20malware%20and%209%2C476%20benign%20applications%20from%20the%20Drebin%20project%20as%20shown%20in%20Fig%202