Malware Classification through Python Machine Learning

Author: Amjad Abu-Mahfouz, amjada@my.yorku.ca

York University, Toronto Ontario, Canada, enquiries@cse.yorku.ca

Abstract: The android platform is a vulnerable platform, and on its own without any protection it is susceptible to malware indefinitely. The most massive vulnerability comes from the Google Play Store itself.

ZDNet with the collaboration of NortonLifeLock have conducted a study on the Google Play Store and have concluded that it is the primary source of malware on the android platform by far, totalling almost 70% of all malware installations [5].

Their research analyzed android applications installed between June and September 2019; 34 million APKs were analyzed during this period across 12 million devices, the results showed that 10% – 24% of the 34 million APKs analyzed were considered dangerous or unwanted [4].

The most troubling statistic found was the ratio between the massive amounts of legitimate APK installations and malicious ones being only 0.6%. This leads to a consensus that the Google Play Store is a secure platform, ultimately ignoring the fact that it is by far the most unsecure attack vector for malware installations on the android platform [5].

This paper will give the reader a background on the important aspects of APK files and machine learning. Then it will describe the project goals and tools used to create a machine learning algorithm used to classify APK files as either benign or malware. Finally, it will describe the project's files and folders along with the results.

1 Background Information

1.1 APK Files

An APK file is basically a type of zip file that follows a certain structure or blueprint and contains the files that make up the android application (see Fig 1.1). It is important to understand the anatomy of the APK file to attain a better understanding of these kinds of files in order to distinguish them from the malicious APKs that deviate from this standard APK structure. The normal APK file structure contains the following key files and folders described below [2].

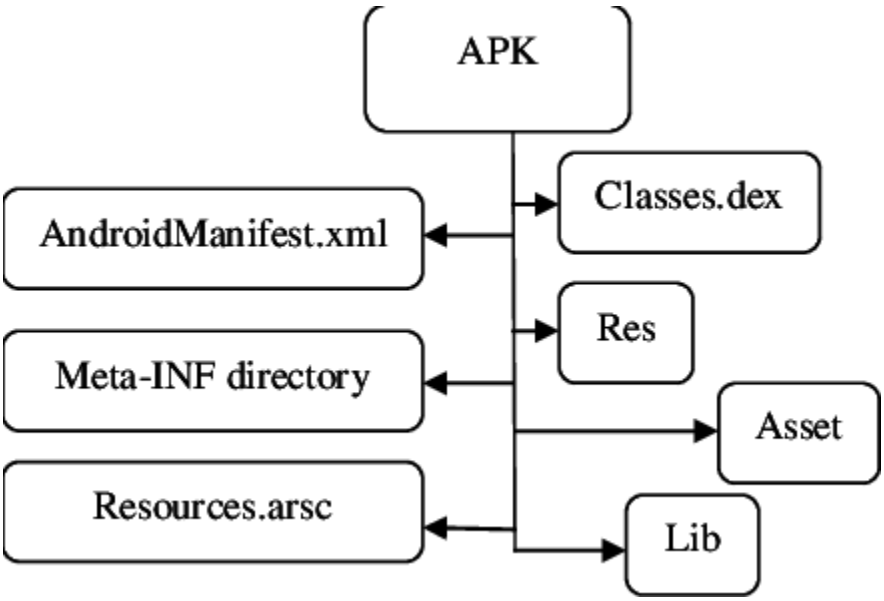

Fig 1.1: this image of the APK structure was taken from research gate website.

Classes.dex:

These files are important and contain compiled code that is used to locate and initialize other program files for the APK.

> The DEX file format and the android OS were made by Google, in the android OS is a module called Dalvik virtual machine which interprets the DEX file code. Regular Java applications could be translated into Android applications by having them indexed into a DEX file; sometimes APKs have more than one DEX file and that is perfectly normal due to there being a limit of 65,536 methods, libraries, frameworks, etc... that can be referenced in a single DEX [1][2].

Res/ :

This folder contains much of the XML files along with the image resources used throughout the APK.

> The XML is used to describe the layouts and where the resources are supposed to be put, and during compilation these xml resources (in res/ folder) are processed into a more compact binary version and would not be readable unless removed from the entire APK folder.it is important to note that the image resources are divided into their own folders based on the image resolution, screen size, and even different languages [2].

Resources.arsc:

This file contains some of the APKs resources that have been compiled in such a way to make them smaller. The file itself is not compressed when it is within the APK zip archive, this is done to achieve faster resource access times [2].

AndroidManifest.xml:

This file is like the other XML resources and will contain a mapping of requirements, libraries, methods, specifications, etc.… this is a mandatory file that all APKs must have and is used by the android OS, Google Play, and the android build tools [3].

> The android manifest has many uses, some of which include building the APK application and its components using the XML, referencing it throughout the android OS and Google Play, setting permissions for the application to access sensitive information and other apps, and the requirements of hardware and software to run the APK [3].

Libs/:

This folder contains the library files that the APK needs to run. Note that any native library ".so" file will be put into its own separate sub-folder named after the CPU architecture (x86_64 for example) [2]. During installation, these files are copied into the "/data" partition resulting in duplicate libraries and memory space wasted [2].

Assets/:

This folder is used to hold any arbitrary assets that are not used by the android-type resources, and is basically a folder for storing any third-party assets such as videogame textures, models, etc... [2].

META-INF/:

This folder contains various files which are a collection of all file names in the APK along with their signatures and the hashes used for encryption; this is much like the signature verification that is used for websites with certificates [21].

> It is good to note that these are compared to the uncompressed versions saved in the archive, and they are checked one by one meaning that no matter the compression level, no resigning needs to be done [2].

1.2 Machine Learning:

Machine learning is a subset of artificial intelligence, this technology uses algorithms to build mathematical models, and these models require inputted datasets to make predictions for future data [6].

Machine learning algorithms have many useful applications such as image and speech recognition, email filtering, recommended suggestions, etc... [7].

There are 3 main branches of machine learning: supervised, unsupervised, and reinforced. Each of these have different approaches to machine learning and different uses, and are explained in the section below [6].

Supervised learning:

Supervised learning requires human “baby sitting” and carefully preprepared datasets with outputs that are selected for the purpose of training the algorithm. This is done in order to make a model that best predicts the output given the input dataset [6] [8].

Once the model has been trained with human supervision on the training dataset, it is tested with the test dataset to see if it could map the inputs to the correct outputs [6].

Unsupervised learning:

As the name suggests, there is no human intervention or preprocessed and labelled datasets. This machine learning algorithm is fed a dataset that has not been labelled, categorized, or classified and it tries to associate certain patterns with certain outputs [9] [6]. The overall goal of unsupervised machine learning is to restructure chaotic data into more organized groups with similar patterns [9].

Reinforced learning:

This kind of learning involves rewarding good behaviour/actions and penalizing undesirable actions. These kinds of machine learning algorithms give automatic feedback in the form reward points and the overall goal is for the learning agent to interact with its environment and maximise the reward point total to achieve the best fitting model [6].

1.3 APK analyzers:

There are various tools that can be used to scan APK files and retrieve information, they are called APK analyzers. These range from imported libraries to be used in line or external ones that stand alone or on the web that output the results in a specified folder.

Below are some of the APK analyzers considered for this project.

Android Studio APK Analyzer:

This tool comes as a part of the android studio IDE (as a toolbar option) and has a command line version with the “apkanalyzer” command. This analyzer is one of my favorites due to its fast speed and ease of use.

When used in the android studio IDE, it displays the information of the APK in a GUI and is neatly organized with much of the work being done automatically under the hood rather than manually by hand. The information analyzed includes the APK size and the size of its component files, the structure of the DEX files, and it even can compare 2 APKs side by side. Ultimately, I opted not to use this one because even though it was very useful and fast in analyzing, automating it with the command line version would have been messy [10].

SISIK online APK Analyzer:

This APK analyzer was perfect in every way imaginable except for one: it was online and could not be used in-line with the code. This website is very easy to use and understand, all the users must do is drag an APK file into the drop box and the website will display and label all the relevant information in seconds [11].

Before, I remarked that the Android Studio IDE analyzer was fast, but this one is around the same speed if not faster. The APK analysis information is vast and has everything from requested permissions to certificates and signatures, and even though the DEX files were not analyzed with the SISIK “apk-tool”, the same website has another page and tool for doing just that called “apk-dump”. Once again, this APK analyzer was not selected due to it being online, meaning the APK’s features would have to be extracted by hand rather than automated by code.

2 The Project

2.1 Project goals:

In this project, the goal is to analyze various APK files and try to assign them to either one of two groups: Malicious or Benign.

Several hurdles need to be crossed to complete this goal, first of which is scanning the APK files to assess the APK's contents and determine which group that APK falls under. Luckily there exists a set of tools and libraries that does the exact things required for this project; the development environment chosen was python since it is ranked as the top language for machine learning due to its comprehensive libraries and tools for this very purpose.

2.2 Machine Learning:

The type of machine learning that best suits the project goal is the supervised machine learning. Supervised machine learning is comprised of two main categories of algorithms: Classification and Regression [12].

Classification algorithms are exactly what is needed for this project, they are used to predict categorical data in the sense of looking at inputs and assigning them to the proper output *or category* [12]*;* for example: [Yes/No] , [0/1], [Benign/Malware]. I will go into more depth on these when describing the building process.

Classification algorithms need a carefully processed and labelled dataset to work effectively; to acquire this dataset I would need a large sample of malicious and benign APKs, an APK scanner to extract the features from the APK, and lastly the best suited classification machine learning algorithm to scan through this dataset and create a good fitting model.

2.3 APK Analyzer:

Amongst the various APK analyzer modules for python, the one called androguard had the most depth and complexity with many options that deal with extracting feature information from APK files [13]. At first it seems simple and easy with the "AnalyzeAPK" method that returns only 3 objects, but the complexity quickly escalates with each of the objects having a wide array of analysis method-calls to choose from.

I chose this python module to be my APK analyzer for the project due to its wide selection of analysis related methods and it mainly being on the python platform which would interface with the machine learning modules easily compared to the other analyzers.

2.4 APK Feature Extraction:

Androguard's "AnalyzeAPK" method takes an APK's file path as its parameter and does the analysis on it. Once the analysis is finished, the results are outputted into three objects: a, d, dx; an example of this analysis code would be < a, d, dx = AnalyzeAPK("./myAPK.apk") > [13].

The three objects that are returned correspond to: the APK as a whole (a), the array of DalvikVMFormat objects that represent the DEX files located in the APK folder (d), and lastly the Analysis object which summarizes all the DEX files into one object and contains special classes for quick information extraction (dx) [13].

It is important to note that the command takes a long time to complete and siphons a sizeable chunk of the system's resources, the major contributor to this is the array of DEX objects (d) which can be omitted by replacing the "d" with an underscore to signify that you want to skip this step.

< a, _, dx = AnalyzeAPK("./myAPK.apk") >

After this change, the systems resource use dropped significantly along with the time it takes to finish an APK scan/analyze; for APK files with a size of 150+ megabytes, it took around 5 minutes, however after omitting the array of DEXes (d), the analyze time came to 2 minutes or less.

Originally the plan was to extract only the important features from the APK such as the ones that heavily influenced the classification, however after going through code examples I decided to extract as many relevant ones as possible and have the machine learning algorithm decide which ones were important to the classification; this resulted in a more dynamic approach which works with a broader scope of inputs [31].

2.5 Extracted Features

The features that were extracted and their relevancy to the classification are described in the sections below.

Apk size: this feature is for the size of the APK and is the staple of all outsourced datasets that were obtained. At first glance, there is a common distinction between the benign and malware APK files and their file sizes; all of the malware samples obtained had a much lower file size then the benign ones.

dex size: just like the APK size feature, this one is also common in all of the datasets and roughly describes the size of the APK in terms of methods and classes.

file counter: this feature is also common in the datasets and represents the number of files present in the APK

class counter: this feature represents the nuber of classes used in the APK's code.

method counter: much like the class counter, this feature tells the number of methods used in the APK's code modules.

dynamic counter: this feature is for the dynamic code loading; this allows for an application to load and execute code that is not part of its initial static code. These modules can be loaded during runtime and from remote locations [23].

reflect counter: reflection in java is used for the inspection of classes, fields, interfaces, and methods during runtime; the reflect does not require knowledge of these things during compilation and could be used maliciously to inspect the android system [24].

Device version (target/min/max): these extracted features store the boundaries of the APK's version in order to run properly. The malicious APKs usually have a much wider range.

SMS permissions (send/delete/interrupt): these features are for the instant messaging permissions that the APK is granted. Giving this kind of access and permissions to the phone's instant messaging is not usual among legitimate APKs and has many malicious uses [27].

Device id: this field is personal due to it being the phone's unique identifier much like a MAC address; it is mainly used by advertisers to track user's across google play [25].

Sim country: this field is also personal as it can be used to find the device's location. Specifically, it tells the country code of the sim provider and is listed as a potential threat on the android developer main site [26].

Installed packages: through the use of android's package manager, a list of the installed packages on the android device could be retrieved; these types of libraries could easily be used for malicious purposes and are listed as potential threats in the android developer site [27].

Subprocesses: using android's process module, attackers could get information about a parent process, access it's I/O stream, and even kill it [27].

JNI: Java Native Interface is a powerful tool that allows java code in a java virtual machine to run and execute code and libraries from other languages. Through the use of native libraires, one could also create and manipulate java objects, call methods and classes, and perform checking during runtime [28].

Device permissions (many): various permissions that the user grants to the APK are collected. The malicious APKs tend to have a more intrusive permission signature list [27].

Receiver count: when various events occur in the android system such as booting up or switching to airplane mode, broadcasts are sent based on the receiver list. if an application is registered as a receiver, it would receive these broadcasts even if it is not running [27].

Activity count: an activity is a single thing that the android user does at a time. Each activity is associated with its own window; the malware APKs tend to have less of these from the analyzed dataset [27].

Provider count: a content provider is a class that gives access to structured data managed by the application. These providers are used to centralize data so that it could be accessed across multiple applications in a format similar to database queries [30].

Service count: a service is a long-running operation that runs in the background; once initialized, a service may continue running even when switched to another application [27]. Application components could bind to a service in order to perform tasks such as file I/O, network commands, and interacting with providers and all of this is done in the background [27].

Exported activities: if an activity is labelled as exported, it could be called from other external applications. The exported value is usually set to false to limit exposure and secure the application [27].

Picture counts: these are the image resources mentioned in the section above. These are divided into folders based on their size and resolution, and these counters are an extension of the file size counters that provide a more detailed breakdown of the APK structure [27] [2].

is_malware: this feature is the manually inputted one that the entire project relies on. This feature/field is used to train the classifier models to then predict whether an unknown APK is malicious or not.

The APKs were all analyzed using androguard and their features were extracted. The analysis of 592 APKs took around 13 hours when analyzing them one by one, after each analysis a new entry in the dataset was added.

The dataset in question is a csv file named “testing.csv” that contains a database-like table of all the features mentioned above as columns with each APK’s entry as rows.

2.6 Machine learning:

In python, the top library for machine learning is called “sklearn” [15]. This library contains all the machine learning algorithms and tools that will be used in the project including the ones mentioned below.

After the features were extracted, they needed to be fed into a classification machine learning algorithm. To achieve the best results in terms of a classifier model, ensemble learning was used [29]; this is a technique where multiple classification algorithms are combined, in this case, they were ranked against each other in a for loop, and the one with the highest accuracy was selected to be used for the final model. The classification algorithms were picked from JavaTpoint.com, a leading tutorial site for software technologies, more about them is described in the section below.

Each of the classification algorithms take in 2 sets of data representing the inputs (x) and outputs (y); their main objective is to find patterns that link the inputs to the outputs and create mathematical models of the relationship from the training phase. After training, the trained model could be used to predict the outputs for a given input with the accuracy of the predictions proportional to the amount of data fed to it during training.

2.7 Classifier algorithms used:

Decision Tree Classifier:

This algorithm uses trees with two types of nodes representing decisions nodes and leaf nodes that represent the outcomes or endpoints; the tree represents all possible outcomes given the decision nodes [16].

when this algorithm is run, it begins at the root node and recursively divides each node into two based on the features that are most likely to achieve the result (y) and stops once the result has been reached. The feature ranking system looks at each node and gives it a purity percentage, a node is considered 100% impure if this node is split evenly 50/50 and is 100% pure if all the tree’s data belongs under 1 class; the Gini index is used to measure the purity or the probability of 2 randomly selected items being of the same class [16].

Random Forest Classifier:

This is basically an array of decision trees on the subsets of data; it takes the average of all trees in the array to get an accurate estimate [17]. Due to the classification being based on the average of trees, some of which would result in the wrong classification, it is recommended to have a large dataset for better accuracy.

K-NN Algorithm:

This is the K-Nearest Neighbour algorithm, and it works by plotting the features into two separate groups, in our case the groups would be malware and benign; it then takes a new feature in and calculates the Euclidean distance to an arbitrary number of neighbours set by the user (K) [18].

The new feature is assigned to the group with the greatest number of nearest neighbours. This is a lazy learner algorithm [18] because it does not use the training dataset to build a model right away and instead waits until a classification is issued to use the training set.

Gradient Boosting Algorithm:

This classification algorithm that works around the premise of strengthening the weak part of the data to make it more accurate. For example, after a random forest algorithm has concluded the results, the data that has been successfully mapped and classified is discarded while the inaccurate data that is left over is retrained again recursively [19].

The gradient boosting classifier is much like the ADA boosting algorithm, with the difference being that gradient boosting does not recalculate the weights for the features after every round of retraining [19].

ADA Boosting Algorithm:

This algorithm is like the gradient boosting algorithm but after each round of retraining, the final classification is influenced by the votes of the weaker trees [20]; this essentially means that the weaker trees have a heavier influence of the end classification. The trees that do the worst have a higher weight value and more work assigned to them to root out the “hard to determine” classifications.

3 Project Structure

3.1 Project Layout/overview:

The project structure is as follows: the parent folder contains the main python code modules, the androguard directory, along with 2 other folders called MyApk and MyData that store the resources used by the main python modules. MyData contains the csv datasets and trained models while MyApk contains the collection of APK samples which are further divided into folders called Benign and Malware.

3.2 Main Code/Casses:

Classifier.py: This python class has the machine learning algorithms in it and is responsible for training the machine learning models using the dataset. the dataset csv file is read, after which the “is_malware” column is assigned to the variable “y”, giving “x” all the other columns except for the “apk_name” because strings are not relevant to the algorithms.

Since many features were extracted from the APKs, only the important ones with the highest weight need to be chosen for the classifier algorithm. The x and y datasets were fitted into an ExtraTreesClassifier algorithm (basically a random forest classifier algorithm), once the model has been generated it was inputted into the SelectFromModel method; this method returns the features based on weight values from the model. It is good to note that every time Classifier.py was ran, different features were selected based on their importance levels.

The dataset has been divided into the x and y variables and now is further divided into the training and testing x and y variables; the rows in the data set are shuffled randomly. This splitting and shuffling can be done in one line of code by using the train_test_split method from the sklearn.model_selection module.

The features/columns in the “x” variable may contain extremely large numbers, specifically for the APK size column, and to save memory space as well as scaling down the numbers to a more human readable format, the StandardScaler from sklearn’s preprocessing library has been used on the “x” data set; the “y” set did not need to be changed because it only contains one column that has a binary value of 0/1. Using the scaler affected the resulting classifications at the end and will be explained in a later section.

The classifier algorithms were initialized in a list, and then that list was used in a for loop; during this process, each classifier algorithm was fitted with the training sets to create the models, promptly after, the test datasets were used to score the model’s classification accuracy. After training and testing all the classifier algorithms and printing their scores, the highest score is selected and its corresponding classifier is chosen as the top classifier (see Fig 3.1 and 3.2).

Lastly, the top classifier model and the important features model are both written and saved into files with a “.sav” extension in the MyData folder. This can easily be done through the use of python’s pickle module which lets you serialize objects such as files and perform manipulation on them such as read and write.

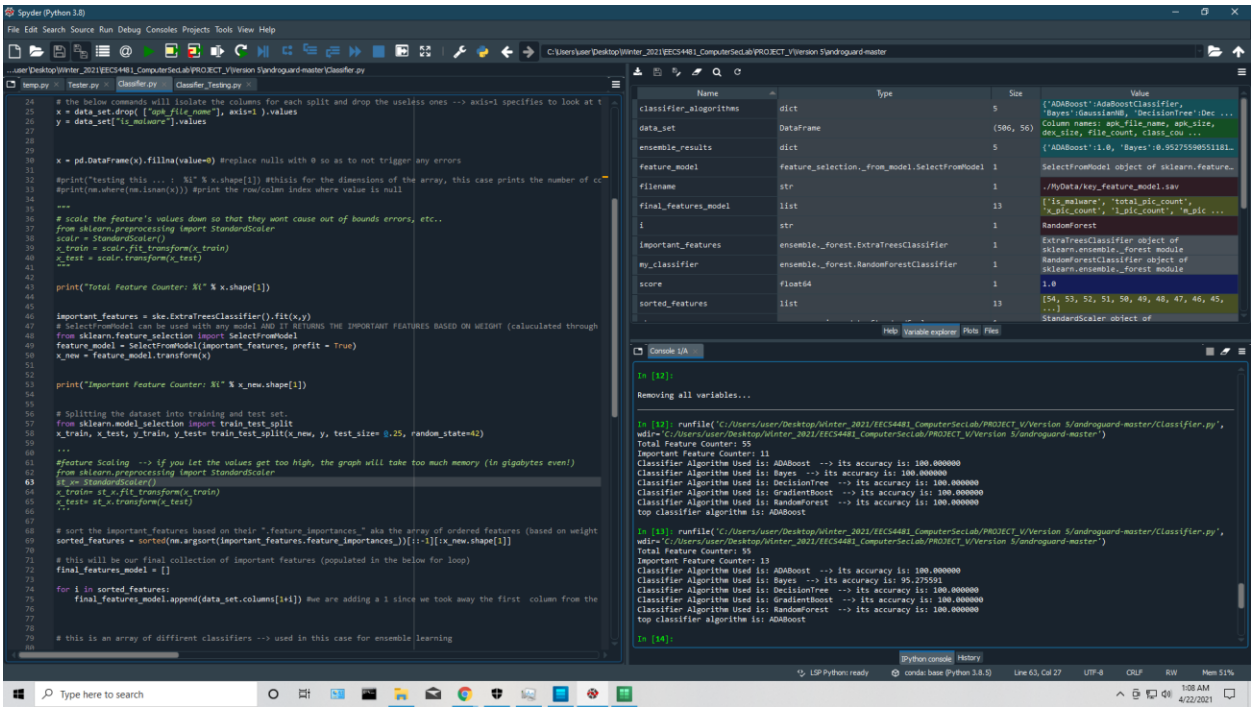

Fig 3.1: my dataset was used to calculate the accuracy of the classifier algorithms. The top output is the classification with scaling, the bottom is without the scaler.

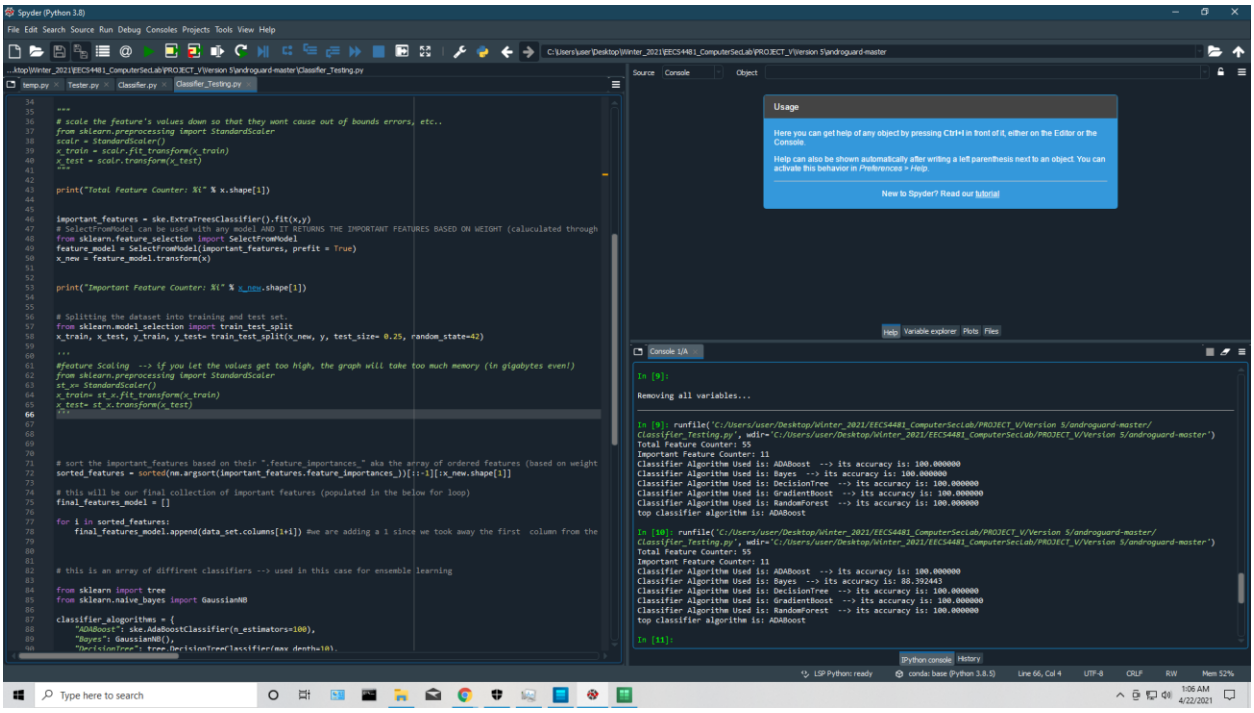

Fig 3.2: a third-party dataset was used to score the accuracy of the classifier algorithms. The top output is the classification with scaling, the bottom is without the scaler.

FeatureFetcher.py: This python program is used to extract the feature set from the APK files and at the end append them to a csv file. To accomplish this, the androguard's analysis object needed to be imported for APK analysis and feature extraction as well as python's zipfile and os modules to examine the APK's file contents and size.

Most of the features pertaining to malicious actions such as deleting SMS messages and tracking the phone user's location were extracted by analyzing the DEX analysis object (dx). For example, on order to find out if the APK in question has the capability of sending SMS messages on its own, one would have to look through the methods of the dx object and search for the specific imported library that would give the capability of doing so; in this case, the library being "Landroid/telephony/SmsManager".

Majority of the features extracted were done through searching matching methods, a few features were extracted through using getter methods provided by androguard; some of these include the file counter that is retrieved from counting the number of lines returned by the get_files() method of the APK object (a). The class counter was acquired by using the get_classes () method of the dx object then counting the length of the return, the same was done for the method counter. Androguard provides many other useful analysis methods to extract features such as the get_target_sdk_version() and get_permissions().

At the end of this program, the extracted features were saved into a csv file called "testing.csv" located in the "MyData" directory of the root folder.

Tester.py: This file is like the control module that connects the FeatureFetcher.py and Classifier.py together. The user is prompted to enter a few inputs before it begins running the analyzer. The first input tells the program what kind of scan to perform and takes either 0 or 1 and this value will control which csv file the FeatureFetcher.py will enter the extracted features in.

If they enter 0, the FeatureFetcher.py will enter the extracted features into MyData/testing.csv and this is the normal file used for the training dataset. if the user enters 1 then FeatureFetcher.py will overwrite the extracted features into MyData/extracted_features.csv; this file is meant to hold one APK's features at a time and is used to test the trained model.

The next input takes in the APK file's name, the file name must have the ".apk" extension included and must be in the "MyApk" folder.

The last input is the malware identification number, entering 0 would label the APK as benign and 1 would label it as malware. If the user entered 1 as their first input, then this input does not matter as only the APK's features would be used to test if the APK is malware, and the malware identification number would not be used at all.

After the inputs, the important feature model as well as the trained classifier model are loaded from the "MyData" folder, where Classifier.py saved them in. the extracted_features.csv is also loaded and the data is scaled down along with all nulls being replaced with zeros to avoid any errors. Lastly, the extracted features are fitted to the important features model to isolate the key features as they need to match the ones from the trained top classifier model. Finally, the extracted and processed features are put into the top classifier model and the predict() method is ran on it in order to estimate the classification of the APK (see Fig 3.3).

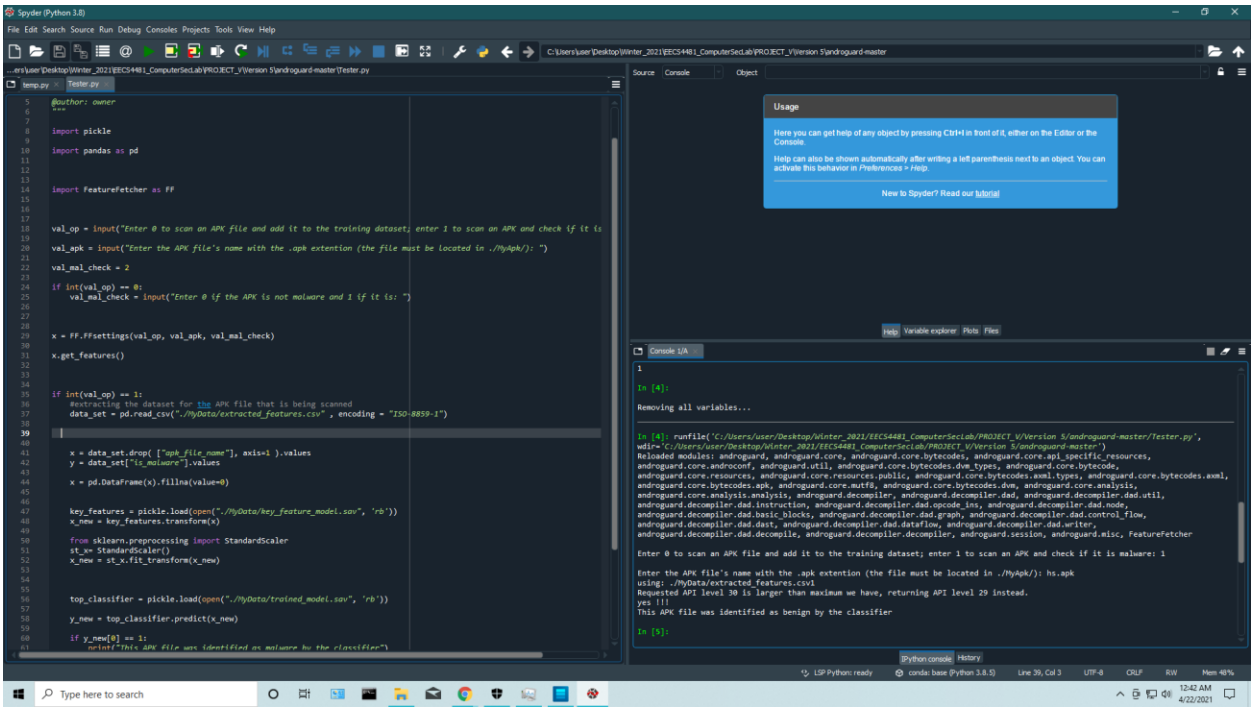

Fig 3.3: Tester.py correctly guessed a classification of an APK that is not present in the dataset.

4 Results:

I have used not only my own dataset but others as well to test the validity of the classifier. My dataset had more features initially in comparison to the some of the other datasets I have tested, that is until the important ones were ranked and the others discarded; on the other hand, my dataset had far fewer entries, around 600 when compared to the third-party ones I downloaded which sometimes numbered in the tens of thousands.

When the outsourced datasets were used in the training model, the resulting trained model had an accuracy in the mid to high 90s (95-99%), while using my own dataset that contained much fewer entries but a larger array of features resulted in mostly perfect accuracy with the exception of the Bayes algorithm which had about 98-99% accuracy. I believe that the high accuracy score of my own dataset could be attributed to the number of features that it contains along with the additional step of ranking the features and selecting only the most impactful ones. Another notable observation occurred when I scaled down my feature's values using the standard scaler, after doing so the accuracy seemed to have slightly improved; this was done to reduce the system's resource load when storing very large numbers but it seemed to have an inadvertent effect of increasing the prediction accuracy of the trained model.

An interesting thing to note was the important feature ranking using the extra trees classifier; every time the "Classifier.py" was ran, it selected different important features and in different amounts. This result was due to dataset being shuffled randomly prior to splitting up into the training and testing sets.

References:

1. ReviverSoft. (n.d.). File extension search. Retrieved April 22, 2021, from https://www.reviversoft.com/en/file-extensions/dex#:~:text=in%20this%20case%2C%20.-,dex.,file%2C%20the%20program%20launches%20automatically.
2. Kaliciński, W. (2016, May 25). #SmallerAPK, part 1: Anatomy of an APK. Retrieved April 22, 2021, from https://medium.com/androiddevelopers/smallerapk-part-1-anatomy-of-an-apk-da83c25e7003
3. App manifest overview : Android developers. (n.d.). Retrieved April 22, 2021, from https://developer.android.com/guide/topics/manifest/manifest-intro?gclid=Cj0KCQjw6-SDBhCMARIsAGbI7UjLqamCn5tMfn9Tsu3MVyLwdLHeWc0VBHQ7fZ_fSCRXajTNVoihJVIaAgZfEALw_wcB&gclsrc=aw.ds
4. Humphries, M. (2020, November 12). Google Play Store Is Main Distributor of Malicious Apps, Study Reveals. Retrieved from https://www.pcmag.com/news/study-reveals-googles-play-store-is-main-distributor-of-malicious-apps
5. Cimpanu, C. (2020, November 11). Play store identified as main distribution vector for most Android malware. Retrieved April 22, 2021, from https://www.zdnet.com/article/play-store-identified-as-main-distribution-vector-for-most-android-malware/
6. Machine learning tutorial: Machine learning with python - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/machine-learning
7. Applications of machine learning - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/applications-of-machine-learning
8. Supervised machine learning - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/supervised-machine-learning
9. Unsupervised machine learning - JAVATPOINT. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/unsupervised-machine-learning
10. Analyze your build WITH APK Analyzer : Android developers. (n.d.). Retrieved April 22, 2021, from https://developer.android.com/studio/build/apk-analyzer
11. Sixo online APK Analyzer. (n.d.). Retrieved April 22, 2021, from https://www.sisik.eu/apk-tool
12. Classification algorithm in machine learning - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/classification-algorithm-in-machine-learning
13. Getting started¶. (n.d.). Retrieved April 22, 2021, from https://androguard.readthedocs.io/en/latest/intro/gettingstarted.html
14. Androguard package¶. (n.d.). Retrieved April 22, 2021, from https://androguard.readthedocs.io/en/latest/api/androguard.html#
15. Learn. (n.d.). Retrieved April 22, 2021, from https://scikit-learn.org/stable/index.html
16. Machine learning decision tree classification algorithm - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
17. Machine learning random forest algorithm - javatpoint. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/machine-learning-random-forest-algorithm
18. K-Nearest neighbor(knn) algorithm for machine learning - JAVATPOINT. (n.d.). Retrieved April 22, 2021, from https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
19. Brownlee, J. (2020, August 14). A gentle introduction to the gradient boosting algorithm for machine learning. Retrieved April 22, 2021, from https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/#:~:text=Gradient%20boosting%20is%20a%20greedy,the%20algorithm%20by%20reducing%20overfitting.

20. Desarda, A. (2019, January 17). Understanding AdaBoost. Retrieved April 22, 2021, from https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe
21. Signed APK Meta-inf Files-digests,signature and certificate - Working tips. (n.d.). Retrieved April 22, 2021, from https://sites.google.com/site/jamestu6166workingtips/gm-git-command/signed-apk-meta-inf-files-digests-signature-and-certificate
22. Classifier COMPARISON¶. (n.d.). Retrieved April 22, 2021, from https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
23. Chandora, K. (2019, October 13). Dynamic code loading in Android. Retrieved April 22, 2021, from https://kalpeshchandora12.medium.com/dynamic-code-loading-in-android-dea83ba3bc85
24. Ashtikar, R. (2021, March 22). Using reflection in android. Retrieved April 22, 2021, from https://www.haptik.ai/tech/using-reflection-in-android/
25. What is a device id? Why are advertising ids important? (n.d.). Retrieved April 22, 2021, from https://www.adjust.com/glossary/device-id/
26. TelephonyManager : Android developers. (n.d.). Retrieved April 22, 2021, from https://developer.android.com/reference/android/telephony/TelephonyManager
27. Documentation : Android developers. (n.d.). Retrieved April 22, 2021, from https://developer.android.com/docs
28. Introduction. (2002, November 12). Retrieved April 22, 2021, from https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html
29. Polikar, R. (n.d.). Ensemble learning. Retrieved April 22, 2021, from http://www.scholarpedia.org/article/Ensemble_learning#:~:text=Ensemble%20learning%20is%20the%20process,%2C%20function%20approximation%2C%20etc.)
30. Android - content providers. (n.d.). Retrieved April 22, 2021, from https://www.tutorialspoint.com/android/android_content_providers.htm
31. Chushu10. (n.d.). Chushu10/apk-static-features-extraction. Retrieved April 22, 2021, from https://github.com/chushu10/apk-static-features-extraction