Machine Learning for Malware Analysis

Machine Learning
A.Y. 2017-2018

Hossam Arafat
1803850

Abstract

This report describes how machine learning methods can be used to classify Android applications as “Malware” or “Benign”, based on a static analysis of the application’s “manifest.xml” file. Specifically, this report illustrates how a Naïve Bayes Classifier and a Support Vector Machine can be used to classify malware and non-malware files. An introduction about the DREBIN dataset is given, then the algorithm and methodology are discussed; namely, how the Naïve Bayes Classifier and SVM were implemented using the Python library Sci-kit Learn “sklearn”. Finally, a comparison is made between both algorithms’ performances.

Background

A Malware is a harmful piece of software that aims to compromise users or systems. Malware files make use of Software and Hardware glitches to gain remote access to a system and cripple it or gain control of it. Malware attacks have been on the rise globally, showcasing an increase of 85% from the last quarter of 2016 to the first quarter of 2017.

Consequently, Malware Analysis techniques are constantly evolving to combat the constant evolution of Malware attacks and to be able to easily and efficiently detect malicious applications, their variants and their specific class or family.

This report aims to implement a machine learning classifier that classifies malware files that infect the Android platform. Two classifiers are implemented using the open-source Python library Sci-kit learn: The Naïve Bayes Classifier and a Support Vector Machine.

Dataset Division

To implement the aforementioned classifier, a benchmark dataset known as DREBIN was used. This data was collected in the form of 123,453 applications with 5,560 malicious samples and 117,893 benign ones. Each application is represented by the characteristics found inside the application's (apk) "manifest.xml" file which details 10 features about the application that can be grouped as follows:

1. Requested hardware components
2. Requested Permissions
3. Filtered Intents
4. Restricted API calls
5. Used permissions
6. Suspicious API calls
7. Network addresses
8. App components
    a. Activity
    b. Service Receiver
    c. Service Provider
    d. Service

This report's implementation considers only 3 features, namely, the Hardware components, Requested permission and Network address. Which are represented in the manifext.xml file as "feature", "permission", and "url" respectively.

Moreover, only the first 7,300 files of the DREBIN dataset were considered. This resulted in 6,974 benign samples and 332 Malware samples. This amount was considered using a trail-and-error approach, as training the model is done on a 6 year-old personal laptop with minimal processing power, and when trying to train either classifier with more samples, the computer throws a Memory Error or stops responding.

Feature Vector Design

To train the classifiers, an input matrix that had 7,300 rows and had its number of columns built in the following manner:

1. Specify the desired features to be considered, this report considers three: “feature”, “permission”, and “url”.
2. Loop over every file in the dataset, extract every possible value this 3 variables could take and group it into a vector.
3. Represent each file as a vector of 1s and 0s depending on the values that were present and absent from set file.

Algorithm Design

To implement the Naïve Bayes Classifier and a Support Vector Machine, 2 design choices were considered.

Firstly, the choice of programming language was key, as a powerful, fast, and efficient language was needed to train the models locally.

Secondly, the language must have enough libraries and packages that supported large dataset manipulation, memory management and machine learning model evaluation techniques built in.

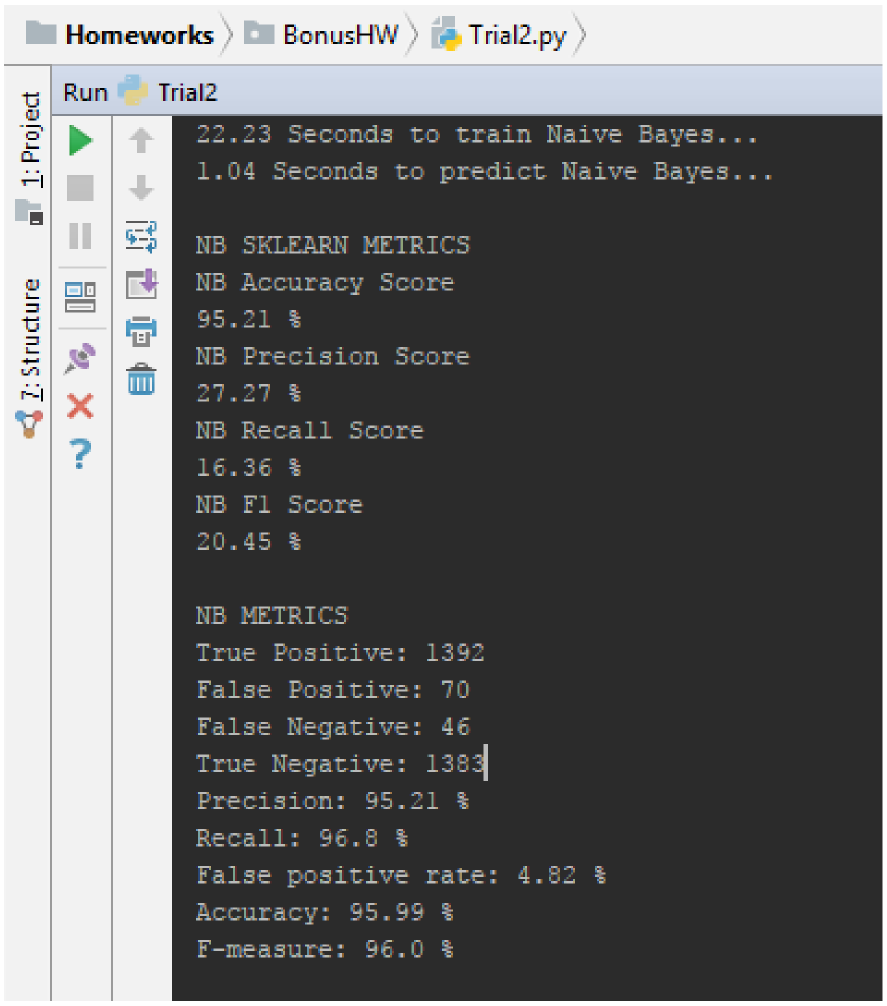
Hence, Python and Sci-kit learn were chosen, as they are open source, easy to use and deploy, have a wealth of clear documentation and a large community of developers backing both projects.

Results & Evaluation

Using Sci-kit Learn’s modules, models were evaluated based on their:

1. Accuracy
2. Precision
3. Recall

Moreover, the Confusion Matrix was calculated which outline the number of true and false positive as well as true and false negatives. The figures below outline the results.



```
22.23 Seconds to train Naive Bayes...
1.04 Seconds to predict Naive Bayes...

NB SKLEARN METRICS
NB Accuracy Score
95.21 %
NB Precision Score
27.27 %
NB Recall Score
16.36 %
NB F1 Score
20.45 %

NB METRICS
True Positive: 1392
False Positive: 70
False Negative: 46
True Negative: 1383
Precision: 95.21 %
Recall: 96.8 %
False positive rate: 4.82 %
Accuracy: 95.99 %
F-measure: 96.0 %
```

Figure 1.1: Naïve Bayes Classifier.

22.23 seconds to train

1.04 seconds to test

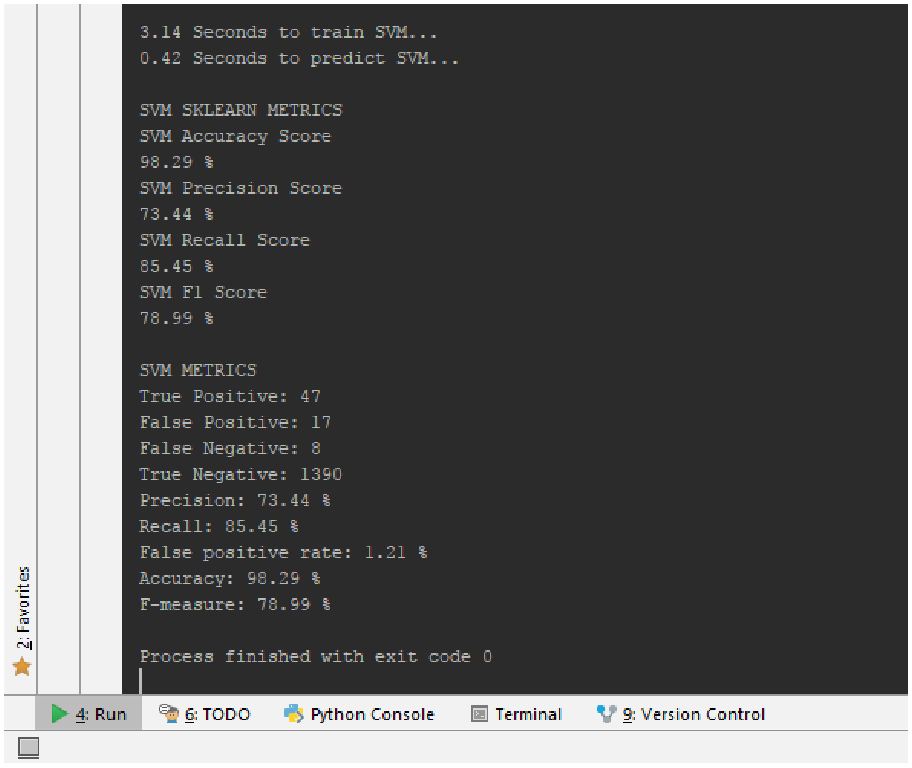
95 % Accuracy

96 % Recall

95 % Precision

Confusion Matrix

1. 1392 True Positive
2. 70 False Positive
3. 1283 True Negative
4. 46 False Negative



```
3.14 Seconds to train SVM...
0.42 Seconds to predict SVM...

SVM SKLEARN METRICS
SVM Accuracy Score
98.29 %
SVM Precision Score
73.44 %
SVM Recall Score
85.45 %
SVM F1 Score
78.99 %

SVM METRICS
True Positive: 47
False Positive: 17
False Negative: 8
True Negative: 1390
Precision: 73.44 %
Recall: 85.45 %
False positive rate: 1.21 %
Accuracy: 98.29 %
F-measure: 78.99 %

Process finished with exit code 0
```

Figure 1.2: Support Vector Machine

3.14 seconds to train

0.42 seconds to test

98 % Accuracy

85 % Recall

78 % Precision

Confusion Matrix

5. 47 True Positive
6. 17 False Positive
7. 1390 True Negative
8. 8 False Negative

Future Improvements

This report recommends trying the algorithm with a larger, more balanced dataset that has a larger number of files with a more balanced ratio between Malware and Non-Malware files. As it is apparent from the results how well the classifiers predict non malware files, as well as how difficult it is to judge their performance on malware files, given the little number of metrics and performance measures available on Malware files in our dataset to begin with.

Furthermore, a more compact, algorithmically efficient representation of the feature vector corresponding to each file is recommended, as it will reduce both time and space complexity.