Android Malware Detection Report

HW1 - Machine Learning class
Sapienza University of Rome

Andrea Fioraldi 1692419

November 18, 2018

Abstract

The detection of malware plays a key role in Cybersecurity nowadays. The amount of malicious program is increasing exponentially over time. Malware programs become cheaper and the attack surface becomes bigger. In the Q3 2016 report [1] alone, 18 million new malware samples were captured. Also, as stated in the Symantec 2018 ISTR [2], more than 24K of malicious mobile applications are blocked every day and, according to FBI [3], more than 4000 ransomware attacks occur every day.

As the number of attacks increased over time, it looks like the number of people working against cyber crimes is not increasing proportionally. From the moment an attack is discovered, it is a race against time to understand how the malware works, what was compromised by the malware and find a fix. When the number of malware is extremely high the effort spent by the people working on analyzing them is not enough.

In this scenario, an automated approach to analyze the behavior of a malware is reasonable. In fact, to address the problem, *Machine Learning* is used with notable results.

In this report, we will discuss an approach based on supervised learning in order to classify an Android application as a malware or not.

1 Introduction

To be able to detect if an Android application is a malware using Machine Learning we considered to analyze a very large dataset. The dataset used is *DREBIN*, from [4], a labeled dataset of circa 123K safe Android applications and 5560 malicious. The description of each program is composed of features belonging to 8 different classes.

1. Requested hardware components (GPS, camera, ...);
2. Requested Permissions (send SMS, access to contacts,...);
3. App components (activities, services, content providers, broadcast receivers);
4. Filtered Intents (Inter Process Communications handled by the sample e.g. `BOOT_COMPLETED`);
5. Restricted API calls (API calls whose access require a permission);
6. Used permissions (permissions effectively used by the application);
7. Suspicious API calls (API calls who allow access to sensitive data e.g. `getDeviceId()`);
8. Network addresses (URLs embedded in the code);

The features from the first four classes are extracted from the `Manifest.xml`, the features from the last classes are extracted from the disassembled Dalvik bytecode.

Inside the dataset, the features are represented using this pattern: `tag::value`. The tags are 11 and they are associated with the features classes in the following way:

Class	Tags
Hardware components	`feature`
Requested permission	`permission`

App components	`activity`
	`service_receiver`
	`provider`
	`service`

Filtered Intents	`intent`
Restricted API calls	`api_call`
Used permissions	`real_permission`
Suspicious API calls	`call`
Network addresses	`url`

2 Preprocessing

We divided the dataset into two parts, one for the training and one for the evaluation. The training set is 2/3 of the original dataset and the evaluation set is the remaining 1/3. We decided also to keep the good/malicious rate the same in the two set.

This division is performed choosing random elements, so running different times the preprocessing phase will affect the final results a bit.

The total number of different features is 6836. Performing some preprocessing we are able to consistently reduce the number of features.

Firstly, we devise to exclude from the training set the features that occur only one time. This choice allows us to reduce the number of features to a more feasible number, 2897. Secondly, regards the URLs class of features, we simplified the meaning of this class considering as a feature only the hostname of a given URL (e.g. http://malicious.site/endpoint becomes only malicious.site).

3 Model

The chosen model is the *Naive Bayes classifier*, a probabilistic model used mainly for text classification. In this model, as explained in [5], a document is viewed as a collection of words and the order of the words in a document is not relevant.

The list of features of an application can be processed like a list of words in a document without losing information, so this model is suitable for our problem.

In our case we divided the applications in two classes c_1 and c_2, respcetively *malware* and *notmalware*.

We considered two variants of Naive-Bayes, the *bernoulli* and the *multinomial*.

The Naive-Bayes model classifies an instance x, represented as a set of features $< x_0...x_n >$, with the class c_m which maximizes the probability $P(c_m|x, D)$, where D is the dataset. The class is computed using the following function:

$$argmax_{c \in C}\{P(c|x, D)\}$$

that is equal to:

$$argmax_{c \in C}\{P(x|c, D)P(c|D)\}$$

And, assuming the conditional independece of the features (words) x_i, is also equal to:

$$argmax_{c \in C}\{P(c|D)\Pi_i P(x_i|c, D)\}$$

In the multinomial variant, this function becomes:

$$argmax_{c \in C}\{\frac{P(c|D)\Pi_i P(x_i|c, D)^{n_{x_i x}}}{P(x|D)}\}$$

Where $n_{x_i x}$ is the number of times the word x_i is in the document x.
$P(x_i|c, D)$ is defined in the following manner:

$$P(x_i|c, D) = \frac{1 + \sum_{d \in D_c} n_{x_i d}}{k + \sum_j \sum_{d \in D_c} n_{x_j d}}$$

With D_c defined as the collection of documents in the training set of class c. The additional 1 and k (the cardinality of the vocabulary) are the solutions to the zero-frequency problem.

3.1 A note about the implementation

We implemented the Naive Bayes classifier in vanilla C++14 to get a significative speed-up over a Python implementation of the training process. This allowed us to experiments different configurations without losing time in training.

4 Experimentation

In order to perform a better classification, we evaluated the performance of training the model using different features classes sets.

The output parameters that we considered fo the evaluation are the following [1]:

- Precision: $TP/(TP + FP)$
- Recall: $TP/(TP + FN)$
- False positives rate: $FP/(FP + TN)$
- False negatives rate: $FN/(FN + TP)$
- Accuracy: $(TP + TN)/(TP + FN + TN + FP)$

The tested combinations of tags to consider are the following:

[1] Legend: TP = true positives, TN = true negatives, FP = false positives, FN = false negatives

Identifier	Chosen Classes
A	`feature permission intent api_call real_permission activity url call`
B	`feature intent api_call real_permission call`
C	`api_call feature intent`
D	`real_permission api_call`
E	`feature intent api_call real_permission`
F	`permission url service feature`

4.1 Bernoulli

In general, the multinomial variant is better than the bernoulli but we tried anyway to confirm this trend also for our case.

The results related to the set A are the following:

- precision: 0.123560280788
- recall: 0.978413383702
- FP rate: 0.312507593983
- FN rate: 0.0215866162979
- accuracy: 0.700027904381

We stopped the experimentation of the bernoulli here due to the very low accuracy. This model seems to classify a good program as a malware too often and it is worst than a dumb model that returns always false.

4.2 Multinomial

Using the multinomial, the results associated with each set are:

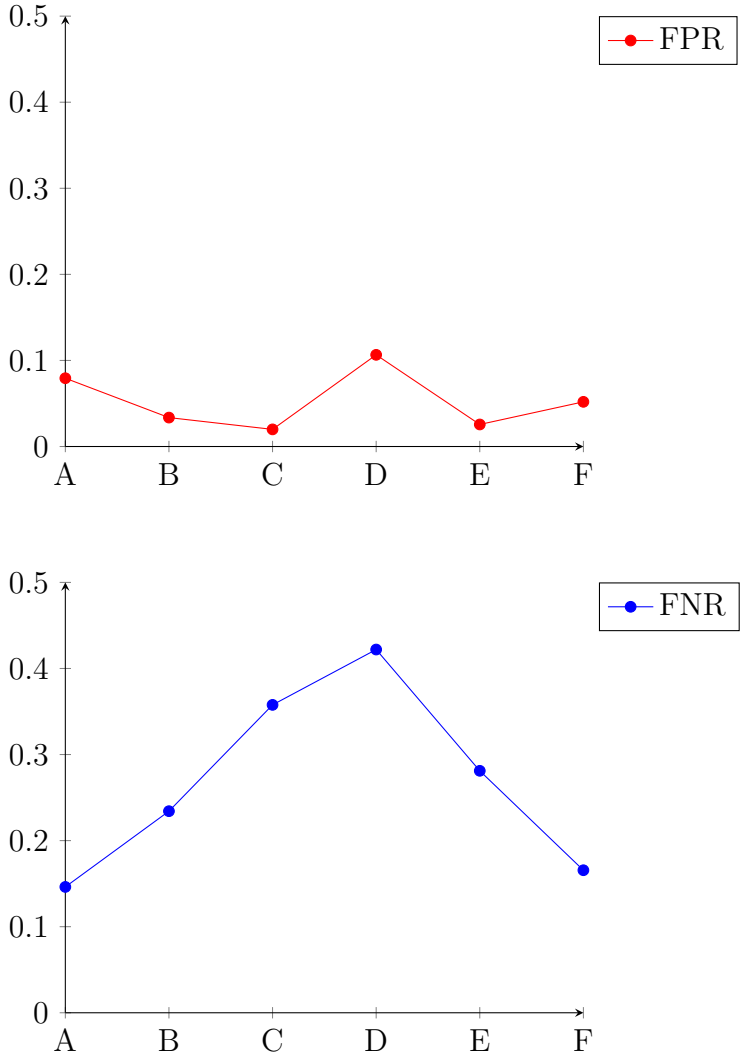
Identifier	**Precision**	**Recall**	**FPR**	**FNR**	**Accuracy**
A	0.3263	0.8537	0.07934	0.1462	0.9177
B	0.5060	0.7657	0.0336	0.2342	0.9577
C	0.5911	0.6422	0.0199	0.3577	0.9654
D	0.1962	0.5779	0.1065	0.4220	0.8798
E	0.5580	0.7188	0.0256	0.2811	0.9633
F	0.4198	0.8343	0.0519	0.1656	0.9431

As you can see, the accuracy metric is not very effective for the evaluation, in fact, the number of *notmalware* samples in the dataset is very bigger than the number of *malware* samples. For instance, even a dumb function that returns always *notmalware* will reach a high accuracy if using this dataset.

For our purpose, the False Positive Rate and False Negative Rate are the important measures. Obviously to perform a good detection FNR must be reasonably low. Regards FPR we do not want to recognize a good program as malicious.

In addition, FPR is a key measure and must be maintained as low as possible for a very important reason that gets out from statistics: legal issues. The FPR rate is very important for many malware scanner software, in fact, they prefer to recognize less malware but do not make mistakes with regular software in order to avoid legal actions by software producers.

Precision and Recall are reported for completeness but they can be easily related to FPR and FNR, in fact, recall is simply 1 - FNR and the precision is higher with low FPR values.



Considering this two measures and looking at the proposed graph we can exclude the sets A and D due to the high FPR rate and C and D due to the high FNR rate. F is similar to A but more interesting due to the lower FPR rate than A, but not enough to be considered. The remaining, B and E, have quite similar results but we choose E because, as said before, the FPR rate must be as low as possible.

An interesting property of E is that the feature classes are probably independents so the multinomial model works well with this choice.

5 Conclusion

The sets of tags tested are only 5 and so a better combination of tags can be found with further investigations. Find the best combination over all possible sets is not trivial and it is not worth the trouble due to the fact that significative better results can be achieved only using a more sophisticated model live Not Linear Support Vector Machine.

As a final thought, we want to discuss the problem that we have addressed itself. Its definition is not simple, in fact, the line that divides the *malware* class from *notmalware* is not well marked. Consider a program that does not perform harmful actions but contains a particular set of features specifically crafted to elude our classifier. Such program that wants to be detected as a malware, is really a malware or not? Eluding a detector can be considered a malicious behavior? This definition problem is an open problem in computer security.

References

[1] "Cybercrime Reaches New Heights in the Third Quarter." https://www.pandasecurity.com/mediacenter/pandalabs/pandalabs-q3/. Accessed: 2018-10-31.

[2] "Symantec - 2018 Internet Security Threat Report." https://www.symantec.com/security-center/threat-report. Accessed: 2018-10-31.

[3] "fbi.gov - Ransomware Prevention and Response for CISOs." https://www.pandasecurity.com/mediacenter/pandalabs/pandalabs-q3/. Accessed: 2018-10-31.

[4] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket.," in *NDSS*, The Internet Society, 2014.

[5] E. Frank and R. R. Bouckaert, "Naive bayes for text classification with unbalanced classes," in *Proceedings of the 10th European Conference on Principle and Practice of Knowledge Discovery in Databases*, PKDD'06, (Berlin, Heidelberg), pp. 503–510, Springer-Verlag, 2006.