

Homework 1

Machine Learning for
Malware Analysis

Michele Ciciolla 1869990

15 Nov 2018

A.Y. 2018-2019

Contents

1. Description of the problem
2. The Drebin Dataset
3. The Malware classifier
4. Description of the algorithm
5. Evaluation of performance
6. Conclusions

1. DESCRIPTION OF THE PROBLEM

This homework has the goal to understand how classifiers work in machine learning problems and to develop a classifier for malware analysis.

There were two possible approaches to the homework:

binary classifier whose task is to detect if an application is malware or not

family classifier that given in input the features of a malware outputs the family it belongs to.

This report shows a possible approach to a Naïve Bayes classifier for malware analysis based on the Drebin dataset which is a public dataset composed by 123,453 benign applications and 5,560 malwares.

Our method has been studied on a smaller part of the Drebin dataset and taking some of the most promising Drebin's features. The performance is attested on 83% with an higher number of false alarms in respect to Drebin.

2. THE DREBIN DATASET

Each application developed for Android must include a manifest file called AndroidManifest.xml which provides features data supporting the installation. They are divided in sets depending on requests and connections they need inside an Android environment.

We had 8 possible families of extracted features shown in the column SET in fig1:

Prefix	SET
feature	S1: Hardware components
permission	S2: Requested permission
activity service_receiver provider service	S3: App Components
intent	S4: Filtered Intents
api_call	S5: Restricted API calls
real_permission	S6: Used permission
call	S7: Suspicious API calls
url	S8: Network addresses

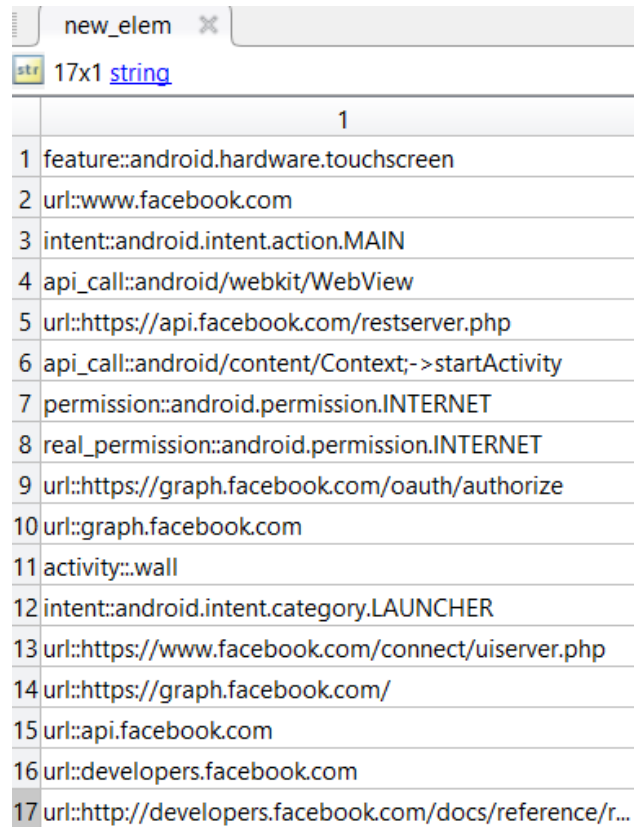
Fig1

Each feature is composed by a prefix and a value, the prefix represents the set the feature belongs to.

Features extracted from an app are stored inside a text file whose name is the SHA1 Hash of the apk, for example, Fig2 shows extracted features from the

"000a067df9235aea987cd1e6b7768bcc1053e640b267c5b1f0deefc18be5dbe1"

hash file:



	1
1	feature::android.hardware.touchscreen
2	url::www.facebook.com
3	intent::android.intent.action.MAIN
4	api_call::android/webkit/WebView
5	url::https://api.facebook.com/restserver.php
6	api_call::android/content/Context;->startActivity
7	permission::android.permission.INTERNET
8	real_permission::android.permission.INTERNET
9	url::https://graph.facebook.com/oauth/authorize
10	url::graph.facebook.com
11	activity::wall
12	intent::android.intent.category.LAUNCHER
13	url::https://www.facebook.com/connect/uiserver.php
14	url::https://graph.facebook.com/
15	url::api.facebook.com
16	url::developers.facebook.com
17	url::http://developers.facebook.com/docs/reference/r...

Fig2

In the next paragraph is explained which features were considered for generating a Naïve Bayes classifier for this problem.

In order to know if an hash file is a malware there were delivered a dictionary table of truth which collects all the malwares that are inside the whole dataset and the family they belongs to, this helped us a lot in calculating performance and accuracy of our system.

3. THE MALWARE CLASSIFIER

The Naive Bayes Classifier (NB) has been applied in diverse fields, especially in language processing and prediction problems. The classifier is called "naive", as it assumes that all features are independent, conditional on the class label.

In this predictive modelling we are interested in modelling a particular process that, for example, given particular features (words) it distinguishes malware from benign sample. The target function $f(x) = y$ is the true function f that we want to model based on our chosen words to looking for.

Our target function is a function that given a sample is capable of categorize it as malware or safe program.

$$f : X \rightarrow V$$

Where X are our sample which are describe by some attributes, and $V = \{Malware, Non - Malware\}$. The attributes of each sample are been taken from the *Drebin* analysis. So the set of attributes is:

- $a_1 = \text{"permission::android.permission.SEND_SMS"}$
- $a_2 = \text{"intent::android.intent.action.USER_PRESENT"}$

as new or unknown attributes, class attributes

Then the class conditional densities can be presented as follows:

$$P(v_j|x, D) = \sum_{h_i \in H} P(v_j|x, h_i)P(h_i|D)$$

And so the Bayes Optimal Classifier for a new instance x :

$$v_{OB} = \arg \max_{v_j \in V} \sum_{h_i \in H} P(v_j|x, h_i)P(h_i|D)$$

Our joint probability model is based on 4 words that were selected according to "*DREBIN: Effective and Explainable Detection*", report over Drebin SVM approach to malware classification.

At page 9 there is an analysis over top malicious features of most common malwares, so we selected some of them inside our classifier.

Malware family	Top 5 features		
	Feature s	Feature set	Weight w_s
FakeInstaller	sendSMS	S_7 Suspicious API Call	1.12
	SEND_SMS	S_2 Requested permissions	0.84
	android.hardware.telephony	S_1 Hardware components	0.57
	sendTextMessage	S_5 Restricted API calls	0.52
	READ_PHONE_STATE	S_2 Requested permissions	0.50
DroidKungFu	SIG_STR	S_4 Filtered intents	2.02
	system/bin/su	S_7 Suspicious API calls	1.30
	BATTERY_CHANGED_ACTION	S_4 Filtered intents	1.26
	READ_PHONE_STATE	S_2 Requested permissions	0.54
	getSubscriberId	S_7 Suspicious API calls	0.49
GoldDream	sendSMS	S_7 Suspicious API calls	1.07
	lebar.gicp.net	S_8 Network addresses	0.93
	DELETE_PACKAGES	S_2 Requested permission	0.58
	android.provider.Telephony.SMS_RECEIVED	S_4 Filtered intents	0.56
	getSubscriberId	S_7 Suspicious API calls	0.53
GingerMaster	USER_PRESENT	S_4 Filtered intents	0.67
	getSubscriberId	S_7 Suspicious API calls	0.64
	READ_PHONE_STATE	S_2 Requested permissions	0.55
	system/bin/su	S_7 Suspicious API calls	0.44
	HttpPost	S_7 Suspicious API calls	0.38

Table 3. Top features for the families FakeInstaller, DroidKungFu, Goldmaster and GingerMaster.

$X = \{\text{word1}, \text{word2}, \text{word3}, \text{word4}\}.$

```
word1 = "call::sendSMS";
word2 = "SIG_STR";
word4 = "permission::android.permission.SEND_SMS";
word3 = "intent::android.intent.action.USER_PRESENT";
```

First one had been selected from S_7 features set of Suspicious API calls because apparently it has an heavy role in suspicious behaviour.

4. DESCRIPTION OF THE ALGORITHM

The Matlab algorithm processes 500 elements from the original Drebin dataset elements, initially divided in $k = 20$ slot. So, the training set for the Naïve Bayes classifier is composed by 20 slots and the test set 5.

The first part of the algorithm is dedicated to the manipulation of the entire dataset and the dictionary table of truth. (Fig3).

```
%-----
% MANIPOLAZIONE VETTORE DEL DATASET
|
filePattern = fullfile('./dataset_1420//','');
files = dir(filePattern);
files2 = struct('name', {files(1:k_dataset).name});
files2(1:2) = [];
dataset = table({files2.name}., 'VariableNames', {'name'});

% estraggo la prima colonna
dataset = dataset(:,1);
% trasformo da matrice ad array
dataset = table2array(dataset);
% converto gli elementi in stringhe
dataset = cellstr(dataset); dataset = string(dataset);
%-----
```

The calculus of the priori probability of malwares in the selected dataset is made up with the `malware_count` function which gives in output how many malicious elements are in dataset according to dictionary table comparing hash names and a list of their indexes inside the dataset (Fig4).

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [num,ml_indexes] = malware_count(dictionary_column,set)
num=0;
% creating an empty vector
ml_indexes = [];
for i=2:length(dictionary_column)
    for j=1:length(set)
        if strcmp(dictionary_column(i,1), set(j,1)) == 1
            num = num + 1;
            % devi salvare gli indici dei malware nel set in un array
            ml_indexes(j) = j;
        end
    end
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Fig4

For each word is calculated the probability given a positive detection using `mal_probability` function which simply compares the feature selected with the set of features of a file, then calculating occurrences and so the probability. The same is done for the probability to get word(i) inside a benign file.

```

pw1_y = mal_probability(word1, dataset, malware_index);
pw1_n = non_mal_probability(word1, dataset, malware_index);
|

```

At the end of this process each word has its associated probability given positive or negative sample.

The main process of this procedure is the analysis and classification of a new element from a new set named Test Set. Each of them receives a

binary classification value equal to 0,1 according to the `naive_bayes` function which gives in output the argmax between values yes and no.

$$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} p(C_k) \prod_{i=1}^n p(x_i \mid C_k).$$

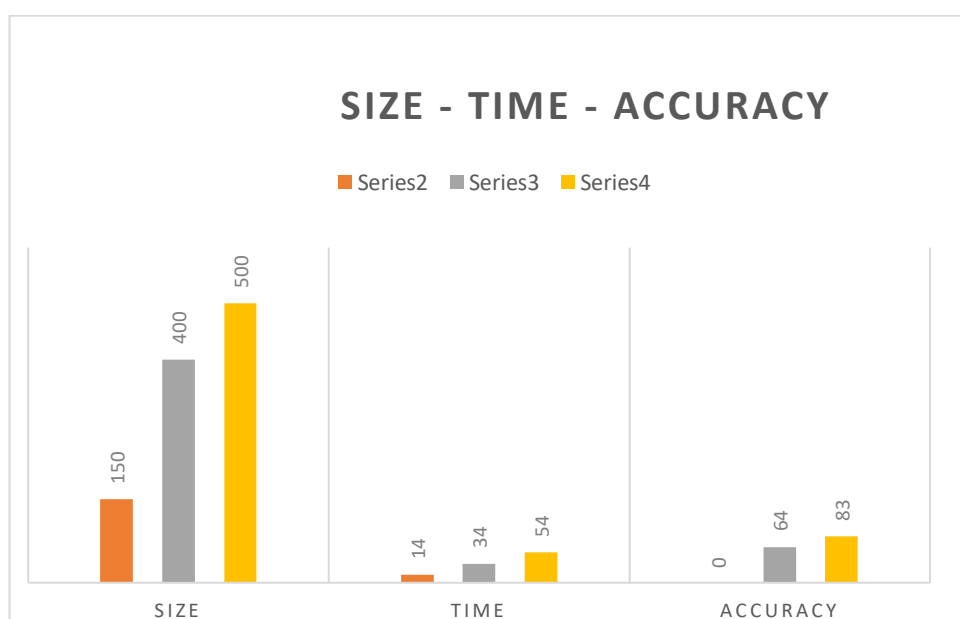
```
yes = pw1_y * pw2_y * pw3_y * pw4_y*0.05;
no =  pw1_n * pw2_n * pw3_n * pw4_n*0.95;

res = argmax(yes,no);
```

We've modelled the possible absence of a word inside an element putting its joint probability to 1 in order to avoid zero products in the calculus. At the end accuracy is calculated based on n° of malwares detected over all of them. There are also considered n° of false positive.

5. EVALUATION OF THE PERFORMANCE

Performance of the algorithm is obviously based on several decisions we made at the beginning of the development. One of these is the size of the dataset. Our testes show that a very poor size of the training set and test set will be processed in few seconds but giving very low accuracy. Enlarging training set of course will give more data to be analysed by the Bayes learning agent. The graph in the figure below explains our result in term of time complexity and size of dataset.



Orange series was our first experiment in term of dataset dimension, at a very poor size of 150 elements, algorithm spends 14' but with 0%

accuracy. At a size of 500 elements divided in 400 for training and 100 for test out accuracy speed up to 83% calculated in 54'.

We will comment over this choice in chapter 6 about conclusions.

The choice of words to search had revealed to be fundamental for the performance of the classifier. Our analysis over the report about Drebrin approach it selected a set of 8 possible features to looking for according to contribution of features sets to the detection of malware families.

```
word1 = "call::sendSMS";
word2 = "SIG STR";
word4 = "permission::android.permission.SEND_SMS";
word3 = "intent::android.intent.action.USER_PRESENT";

% plausible words
% sendTextMessage
% system/bin/su
% getSubscriberId
% lebar.gicp.net|
```

So we extracted the best 8 words according to their contribution in malware detection shown in Table 4.

Feature sets	Malware families																			
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
S_1 Hardware components	✓			✓				✓					✓							
S_2 Requested permissions	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
S_3 App components			✓	✓	✓					✓		✓		✓			✓			
S_4 Filtered intents		✓				✓	✓						✓			✓				
S_5 Restricted API calls	✓				✓											✓				
S_6 Used permissions																				✓
S_7 Suspicious API calls	✓	✓		✓		✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	
S_8 Network addresses			✓					✓	✓	✓	✓				✓	✓		✓	✓	✓

Table 4. Contribution of the feature sets to the detection of malware families.

In particular we noticed that the choice of words to find is the most important parameter that afflicts the performance of a detector. Table 4 was fundamental for this goal because is clear that feature from S2 and S7 are the weightiest for this purpose because they appear many times in malware element.

Choice of features to search is also affecting false positive and false negative performance. In particular our set of words selected gives to the algorithm a percentage of false positive of 5% and false negative of 15%. These results were calculated manually at the end of the procedure just comparing the hash name of the presumed malware with the truth table.

6. CONCLUSIONS

Accuracy of our Naïve Bayes detector at the end of all tests is around 85% with a percentage of false positive of 5% and false negative of 15%. The lack of accuracy of our detector is probably caused by the big prior probability of benign file in our dataset (95%) which really dominates in the calculus of classification output.

Our opinion is that introducing an integer weight w multiplying yes factor in the Naïve Bayes product could raise the chance to improve our accuracy. At this stage w is purely set manually.

```
yes = w*pw1_y * pw2_y * pw3_y * pw4_y*0.05;
no =  pw1_n * pw2_n * pw3_n * pw4_n*0.95;

res = argmax(yes,no);
```

A very raw formulation to get w could checking how many words have been found inside a file and then calculate the weight as follows, obviously giving more focus on instances with 4 words inside.

```
if words_inside == 2
    w = 2.5;
end
if words_inside == 3
    w = 4.5;
end
if words_inside == 4
    w = 9;
end
```

One possible approach to get better could be managing the dataset in order to obtain an equal % of malware and benign file inside it. With a prior probability of 50% our opinion is that the accuracy and performance in general should raise.

Even the choice to select just 4 words is included in this lack of performance because sometimes one of them could be found in a benign file causing a false positive detection.

Selection of words should be larger than just 4 of them and also the selection process should be improved maybe looking for the most popular words inside the selected dataset because the manual choosing of them looking on a paper could be distant from the reality we've in our hand.