Authors & Acknowledgment

Our problem : Malwarology

Strategies & Tactics

Some problems to solve

Machine Learning in a Nutshell

Some (minor) problems to solve

Architecture

Our experiments : Strategy & Tactics

Some results

Automeans

A short conclusion

Machine Learning with Scikit-Learn: Quick Clusterization of a Very Large Malware Dataset

****

(With Open Source Tools

for Fun & Profit)

November 2018

Authors: R. ERRA & S. LARINIER & A. LETOIS
Speaker: R. ERRA

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Authors & Acknowledgement

Authors

- Robert Erra : EPITA-LSE (Speaker)
- Sébastien Larinier : EPITA-LSE
- Alexandre Letois : EPITA-LSE

Acknowledgments

- This research (still in progress) has been partly funded by the French DGA/MI (RAPID Program)
- This was a joint project (ViralStudio) with SEKOIA
- Thanks a lot to Intel for the 4 (marvelous) computers
- Thanks to Marwan Burelle and Mark Angoustures
- And sorry for my "Frenchy" English

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Malwarology : it is the (Data)Science of Malware

1. Analyse Malware
2. Compare Malware
3. Clusterization of a Malware Dataset
4. Classification of new Malware
5. Detection of a Malware
6. Find shared code between Malware
7. Heritage/Lineage/Phylogeny for a Malware Dataset

We consider the following problem :

1. Dataset : **some millions of malware**, the *Malware Data Set* (MDS) or the *𝓑lob*
2. How to compute a **clusterization** of our *𝓑lob*
3. We want to be able (with the same tools) to cluster **hundred of millions** of malware
4. So one of our main objectives is : **scalability** for all computations
5. We present here some strategies & tactics to compute a **clusterization** of our *𝓑lob* . . .
6. using (mainly) **Open Source Tools** : Scikit-Learn + PEtoJson

Our goal today :

To give you information to develop your own tool !

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. **Strategies & Tactics**
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

ML Strategy

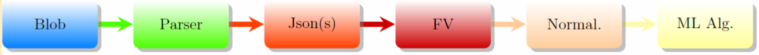
Classical strategy to run Machine Learning algorithms :

- Define and compute a Feature Vector (FV) for each data, this gives you the *Features Vector Set* (FVS)
- Define a (computable) distance between two features vectors, this gives you *similarity* (or a quasi-similarity) between two FVs.
- Run your algorithm(s) on the *Features Vector Set*.

Our Tactics

1. Features Engineering : from a static analysis (parser) we compute FVs
2. Distances : Euclidean distance between two FVs (similarities between two malware)
3. How to choose algorithms ? Kmeans + DBSCAN
4. Scalability ? Kmeans scales well, DBSCAN does not !
5. Kmeans : (unfortunately) you need to know the number of clusters or to follow the Elbow Method
6. Don’t forget : Code and algorithms Optimization (Celery etc.) and always **Normalize** correctly the FVs

Resume : our (first) Computation Pipeline



Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

You will need to solve some problems :

0/ **Maths** : You will probably have to refresh some knowledge about entropy, statistics, etc. to understand your dataset. But also : numerical linear algebra, optimization algorithms etc.

1/ **Features Vectors** (FVs) : You first need good static information. Unfortunately there is no universal representation !

2/ **Parser** : You will need a "very good" parser to compute these FVs.

You will see quickly for example

. . . that some parsers do not like sections with very strange names. So you will soon decide to write your own parser (we have developed two, one is Open Source : *PEtoJson*.

You will need to solve some problems

3/ **Distance** : You will also need a distance metric or a similarity function (at least one) on your FVs to define pair-wise malware similarity but any $O(n^2)$ algorithm is useless.

4/ **Algorithm(s)** : What is the the *"Best Clustering Algorithm"* ? Unfortunately, it does not exist !

- So you will try, for example K-means and DBSCAN, which are good candidates, and you will probably decide, like us, to sometimes use both of them, together.

To go beyond : You will need to solve some other problems

5/ **Deep Learning** : You also probably try to understand how you can use *Deep Learning* algorithm (which seems promising).

6/ **Graphs** : You will probably need/want to use graph algorithms : Louvain is well suited to compute communities (clusters !) in a very large graph (but you will need to compute a very large graph. Well, this means you have have to understand what a t-spanner graph is . . .

You will need to solve some problems

7/ **Data Set** : They become so huge that you will also need to optimize and adapt all algorithms (you will need to parallelize codes you will write).

- You will look for example how to use sparse matrices

8/ **Quality** : Actually you will need some quality functions to understand how many clusters you have (a difficult question) and how valid they are.

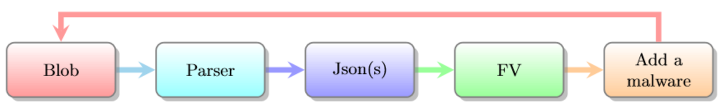
Don't forget that, unfortunately

. . . clustering algorithms always find clusters, even if there are no "natural" clusters in the data.

Your Features Vector Set

. . . depends of your $\mathcal{B}$lob so, if you add new malware, you have to compute (or to complete) again the FVS !

The Computation Pipeline : add (many) new malware



Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Theoretically

. . . Machine learning algorithms help you to learn about a dataset

Practically

. . . they help you to learn about a very large dataset if you have enough CPU power and a minimal *a priori* knowledge of your dataset and fast algorithms.

We were interested by :

- *Supervised* algorithms : we have some information, labels, about each element of the dataset
- *Unsupervised* algorithms : we do not have labels.

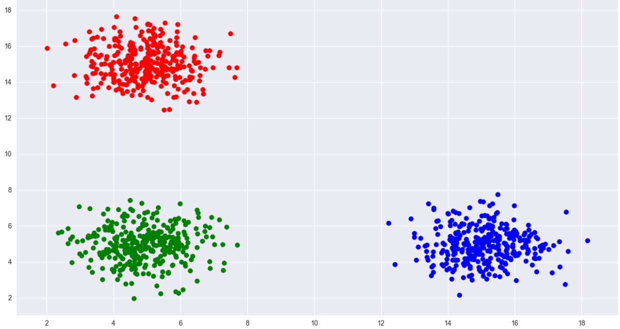
Our tactics

1. We could use *semi-supervised algorithms* : if we would have some *labels* (VirusTotal, Yara rules, AV databases . . .) for a subset of the large dataset
2. We did not have these labeled data (two years ago), well, we have used **unsupervised** algorithms
3. And, unfortunately, the problem is much more difficult.

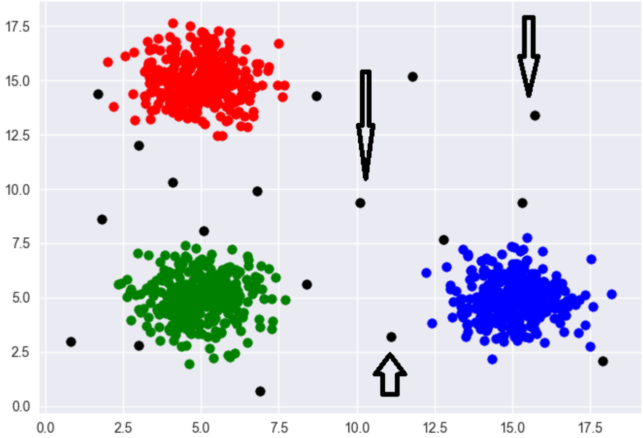
Supervised *vs* Unsupervised ? k : Number of Clusters

Unsupervised	**Supervised**
k : **Unknown**	k : **Known**
No *a priori* knowledge	Training Set (Labels)
To **Classify** future data	To **Understand** data
Kmeans, DBSCAN, . . .	KNN, SVM, RNN . . .

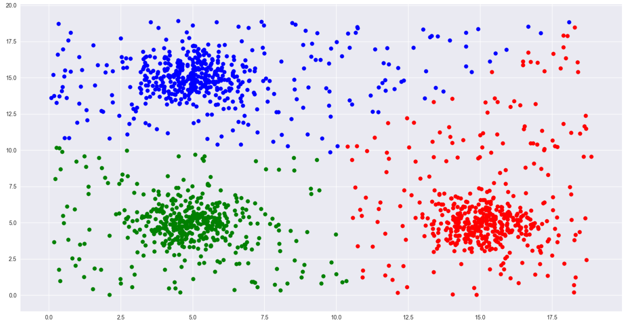
How many "clusters" do you see ? 1/4 (Kmeans)



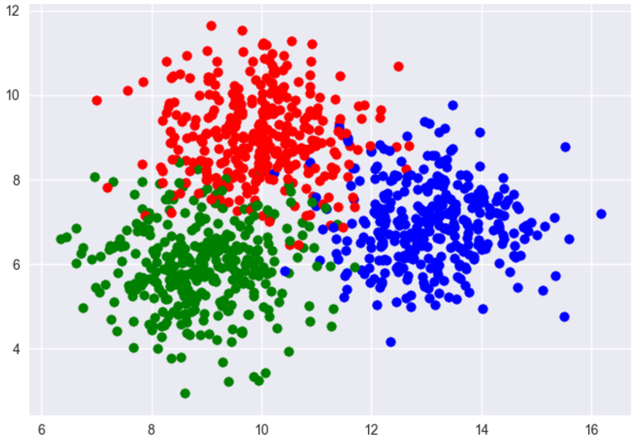
Do you still see 3 "clusters"? 2/4



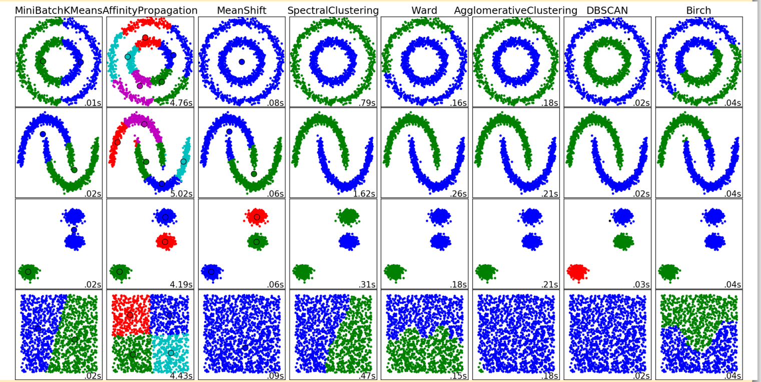
Do you still see 3 "clusters" ? 3/4



Do you still see 3 "clusters"? 4/4



Kmeans is not always the "best"′ choice : DBSCAN is a good option but you have a lot of other algorithms (Don’t forget the scalability)



So : to cluster a set of (numerical) objects . . .

. . . is to group into **meaningful categories** these objects :

- We want objects in the same group to be closer (or more similar) to each other than to those in other groups.
- Such groups of similar objects are called *clusters*.
- When data are labeled : *supervised clustering*.
- It is a difficult problem but easier than the *unsupervised clustering* problem we have when data are not labeled.

Why a clusterization ? To classify !

1. To classify is to choose, for a new data the **best cluster** where to put it.
2. So, when we say we "want" to classify some new malware it means : *we want to choose for a new malware the "best" cluster it owns.*
3. We want to better understand our $\mathcal{Blob}$

A more formal definition

- For the following : let X be the dataset we are studying : $X = \{x_1, \cdots, x_n\}, X \subseteq \mathbb{R}^d$ and $D(i,j)$ the usual Euclidean distance between i and j.
- Let us define $P(X)$, a clusterization of X as a *partition* of X in k (non-overlapping) clusters C_i, *i.e.* :
  - $P(X) = \{C_i\}_{i=1\cdots k}$
  - $\cup\{C_i\}_{i=1\cdots k} = X$
  - For all $i, j : C_i \cap C_j = \emptyset$

Back to our problem

We are interested :

1. to **cluster** our *Blob* (at first : without any labels)
2. and to **classify** a new malware

More exactly :

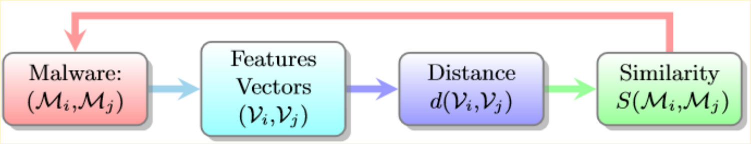
1. We want to compute a quite good *clusterization* of our *Blob* **but** :
   1. in a short time
   2. with little memory.
2. To be able to classify *quickly* some new malware to help human analysts.

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Distance between two FVs gives only a *Similarity* (or worse : a quasi-similarity) between two Malware : $d(X, Y)$

- 1. *Positivity* : $0 \leq d(X, Y)$; 2. *Symetry* : $d(X, Y) = d(Y, X)$
- 3. *Reflexivity* : $d(X, X) = 0$; 4. : $d(X, Y) = 0 \iff X = Y$
- 5. *Triangular Inequality* : $d(X, Z) \leq d(X, Y) + d(Y, Z)$
- 7. *Similarity Inequality* : $d(X, Y) \leq d(X, X)$ if and only if $X = Y$.



Malware: $(\mathcal{M}_i, \mathcal{M}_j)$

Features Vectors $(\mathcal{V}_i, \mathcal{V}_j)$

Distance $d(\mathcal{V}_i, \mathcal{V}_j)$

Similarity $S(\mathcal{M}_i, \mathcal{M}_j)$

Scaling algorithms : the $O(n\log(n))$ Big Data Wall

- Polynomial algorithms are presented as *fast* in quite all CS books.
- But, when n, the size of the dataset, is very large, so, in the real world, we can not use a $O(n^3)$ or $O(n^2)$ algorithms !
- This is the **Big Data Wall** : we are limited with complexity in the order of $O(n)$ or $O(n\log(n))$ algorithms.

Small Powers versus Big Data ?

Small Powers (always) Wins

Store your (sparse) matrix : and save Space & Time

- A sparse matrix is a matrix (generally large) with a lot of elements that are equal to zero : so, store only non zero elements !
- **Not an option**.

A toy example :

$$\begin{bmatrix} 14 & 0 & 19 & 0 \\ 0 & 17 & 0 & 0 \\ 0 & 0 & 18 & 0 \\ 0 & 0 & 0 & 15 \end{bmatrix}$$

```
>>> # csr_matrix(arg1[, shape, dtype, copy])
>>> # Compressed Sparse Row matrix
>>> from scipy.sparse import csr_matrix
>>> from numpy import array
>>> I = array([0,0,1,2,3])
>>> J = array([0,2,1,2,3])
>>> Data = array([14,19,17,18,15])
>>> A=csr_matrix((Data,(I,J)),shape=(4, 4)).toarray()
```

Dataset ? Static Analysis

- Ask to your friends (or to your clients) some millions of malware and use a parser [our parser : ref 4]
- Find a dataset :
  1. theZoo [ref 2] : small but quite good to test your algorithms
  2. Ember [ref 1] : 1.1M binary files :
     1. 900K training samples (300K malicious, 300K benign, 300K unlabeled)
     2. 200K test samples (100K malicious, 100K benign).
     3. only Json files
     4. title : *EMBER : An Open Dataset for Training Static PE Malware Machine Learning Models*

Packers : is it a problem ?

- Many malware are packed with known packers like UPX (used by 98 % of malware packed in our biggest cluster and by 75% on VirusTotal)
- APT are (generally) not packed
- If it's a *homemade* packer, it's used only for a family or a campaign

Not really a problem :

It can be a good idea to unpack because :

1. We can use the Feature Vectors from packed files
2. We can use the Feature Vectors from unpacked files (well, more exactly the ones we can unpack !)
3. We can then combine them to define a new FVS !

Fuzzy hash is a Nutshell

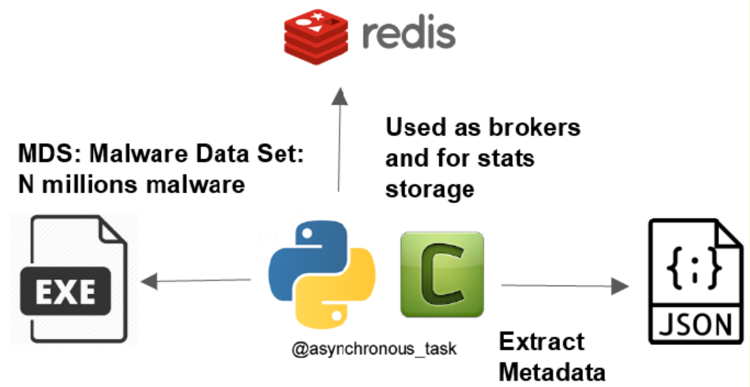
1. When you calculate the fuzzy hash of the two same Import Table but the symbols are not sorted with the same way, you have a 40 % matching
2. ImpFuzzy(CreateProcess + RegWrite + OpenProcess) != ImpFuzzy(OpenProcess +CreateProcess+RegWrite)
3. Two malwares with a very close code (check with Binavi) but one has a big empty section data, the rate matching is 0 : because the block size of the two hashes are different
4. So if a matching rate is equal 0, you have no information about similarities.
5. So for classification, it is better to sort the symbols.

Contents

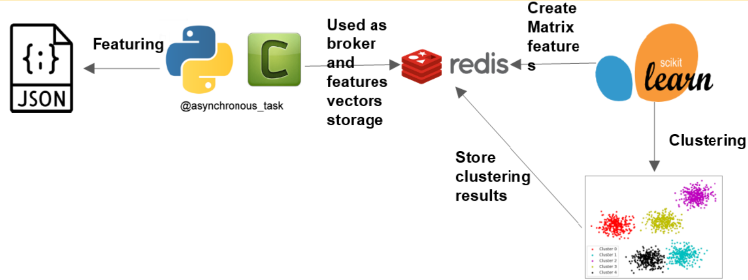
Architecture 0 : Our computers

1. RAM : 64 Go
2. Disk : SSD Intel 240 Gb (SSDSC2BB240G6)/nmve intel 400 Go
    - Processor Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.2 GHz
    - 4 processors
    - 4 x 22 cores = 88 cores
    - Total : 344 cores

Architecture 1 : Extract all Metadata and store


redis
Used as brokers and for stats storage
MDS: Malware Data Set: N millions malware
EXE
@asynchronous_task
{:}
JSON
Extract Metadata

Architecture 2 : Featuring and Algorithms



Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. **Our experiments : Strategy & Tactics**
9. Some results
10. Automeans
11. A short conclusion
12. Future work : next steps

Focus on Metadatas with Static Analysis

We compute (and store) :

1. Hashes : sha256, sha1, md5, imphash, impfuzzy, ssdeep
2. Size, number of sections, size of sections, names of sections [a litlle bit useless], entropy by section, characteristics (hashes etc.)
3. Resources, number of resources
4. Address of entrypoint, section of entrypoint
5. Import Tables, Exports Tables, Callback tls
6. Strings, Certificates
7. Assembly
8. Compilation date
9. Etc.

Classification : some other tactics

1. We compute the full FVS (huge : more than 10^6 dimension for each vector and SPARSE !)
2. But we can use different sub-vectors
3. Algorithms (first idea) :
   - We use K-means algorithm with the chosen sub-vectors, we then have a clusterization
   - We use DBSCAN algorithm with each cluster

Test ? Yes we can

1. We check results with yara rules, Virustotal and "our" statistics
2. We inject labeled malware (TheZoo) and goodware in the dataset and we check their clusterization

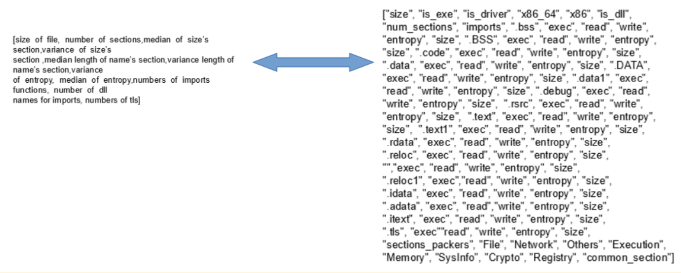
Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. **Some results**
10. Automeans
11. A short conclusion
12. Future work : next steps

Understand your data : some statistics on our $\mathscr{Blob}$

1. Number of sections : 8543138 $\approx 8,5 \times 10^6$
2. Number of different sections : : 2793782 $\approx 2,7 \times 10^6$
3. Number of different section names : 97263
4. Minimal number of sections/malware : 0
5. Maximal number of sections/malware : 90
6. Minimal length of a section : 0
7. Maximal length of a section : 4294966784 (probably a joke because 4294966784 $= 2^{32} - 2^9$)

Features selection ? two examples



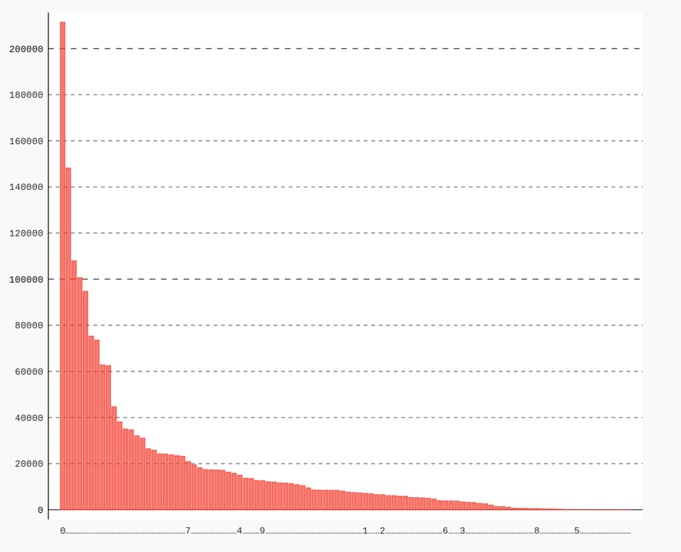
```
[size of file, number of sections,median of size's
section,variance of size's
section ,median length of name's section,variance length of
name's section,variance
of entropy, median of entropy,numbers of imports
functions, number of dll
names for imports, numbers of tls]
```

```
["size", "is_exe", "is_driver", "x86_64", "x86", "is_dll",
"num_sections", "imports", ".bss", "exec", "read", "write",
"entropy", "size", ".BSS", "exec", "read", "write", "entropy",
"size", ".code", "exec", "read", "write", "entropy", "size",
".data", "exec", "read", "write", "entropy", "size", ".DATA",
"exec", "read", "write", "entropy", "size", ".data1", "exec",
"read", "write", "entropy", "size", ".debug", "exec", "read",
"write", "entropy", "size",  ".rsrc", "exec", "read", "write",
"entropy", "size",  ".text", "exec", "read", "write", "entropy",
"size",  ".text1", "exec", "read", "write", "entropy", "size",
".rdata", "exec", "read", "write", "entropy", "size",
".reloc", "exec", "read", "write", "entropy", "size",
"","exec", "read", "write", "entropy", "size",
".reloc1", "exec","read", "write", "entropy", "size",
".idata", "exec", "read", "write", "entropy", "size",
".adata", "exec", "read","write", "entropy", "size",
".itext", "exec", "read", "write", "entropy", "size",
".tls", "exec""read", "write", "entropy", "size",
"sections_packers", "File", "Network", "Others", "Execution",
"Memory", "SysInfo", "Crypto", "Registry", "common_section"]
```

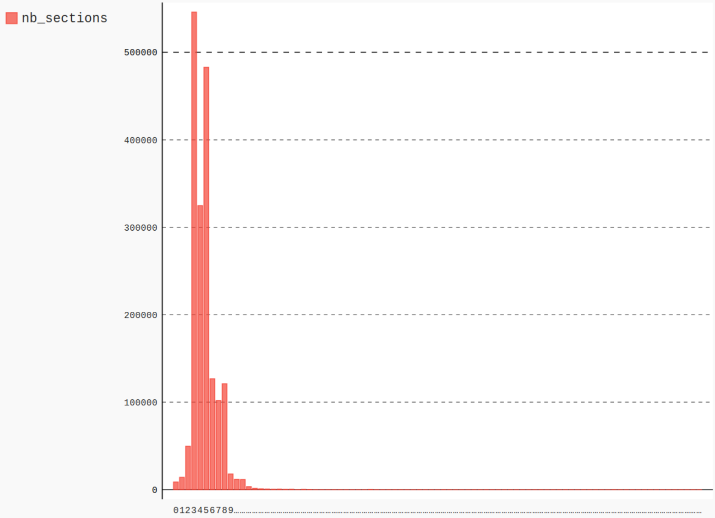
Features selection : two examples

1. The first vector gives a better distribution and separates packed malware or not, version of packers and families malware.
2. The second vector give a "bad" classification with poor matching *yara* rules

Size of Kmeans clusters :



Histogram : Number of sections by malware



(Rank of cluster, number of malware, matching yara %)

The top 10 clusters :

1. (1, 66339, 0.997...)
2. (63, 15920, 0.996...)
3. (90, 23358, 0.992...),
4. (94, 14064, 0.989...)
5. (41, 13955, 0.982...)
6. (46, 11159, 0.981...),
7. (178, 10519, 0.979...)
8. (136, 12452, 0.977...)
9. (109, 4094, 0.977...),
10. (30, 2722, 0.975...)

(Rank of cluster, number of malware,matching yara %)

The 10 last clusters :

1. (236, 698, 0.0014...),
2. (28, 912, 0.0065...),
3. (76, 8555, 0.0086...),
4. (31, 44136, 0.0097...),
5. (75, 6779, 0.0106...),
6. (92, 259, 0.0115...),
7. (241, 289, 0.0138...),
8. (168, 355, 0.0149...),
9. (21, 50298, 0.0160...),
10. (196, 3366, 0.0172...)]

So now if we watch more precisely the cluster 1

On 66339 binaries :

- 3 types of sections ('UPX0','UPX1','.rsrc'), ('UPX0', 'UPX1', 'UPX2') and ('code','text','rsrc')
- All are flagged by Virustotal like UPX packers and we find different versions but Yara rules of the community don't match correctly
- 50000 files have the same Import Table and the same sections of different packed malware .

We then apply the DBSCAN algorithm on the cluster 1 :

DBSCAN finds 10 "sub-clusters", here are the vector results :

```
(Rank of cluster,number of malware)
[(0, 62518),(1, 1167),(2, 521),
(3, 108),(4, 157),(5, 397),
(6, 288),(7, 208),(8, 239),
(-1, 736)]
```

Some results :

We can classify :

- Packers : All different packers between them
- Versions of packers
- Packers families not (always) found by Virustotal and Yara Rules
- Families malware : we "refine" clusters found by K-means with the DBSCAN algorithm, it seems a good idea
- We have injected the Zoo Dataset into the $\mathcal{B}lob$: families are "grouped" correctly

Contents

Automeans : How to avoid the *k*-s by *k*-s

A greedy algorithm inspired by K-means and Louvain

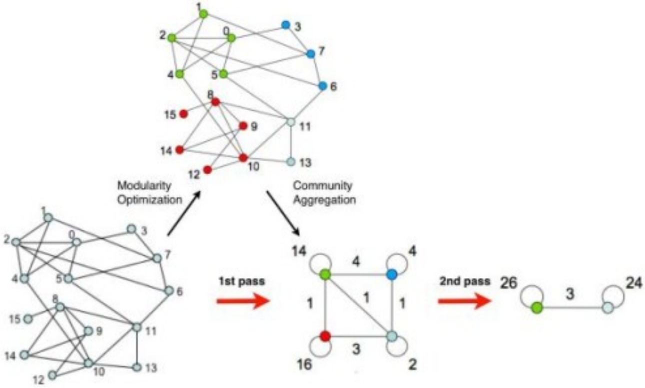
- K-means : a (classical) clustering algorithm that find clusters of similar elements in a set of points
- Louvain : an algorithm that computes communities : groups of similar nodes in a graph
- Automeans is "Inertia-based" : like K-means, with the use of an heuristic
- Automeans uses neighborhood and find the value of *k* dynamically, like Louvain
- Aggregating and shrinking phase, like Louvain
- Can be adapted by changing the heuristic or the neighborhood structure

How to use Automeans ?

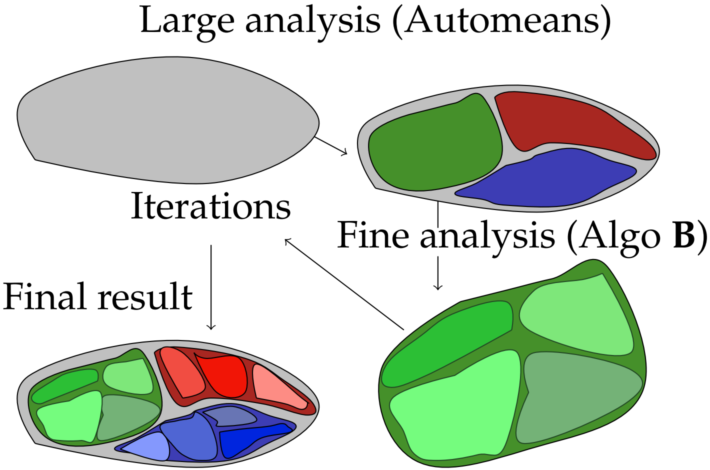
Properties of Automeans

- Can be used to get a good partition on a dataset without previous knowledge
- It is fast and can scale to millions of files with (moderate) dimensions
- Designed to be used as a first pruning algorithm [think to the Elbow Method]
- Time complexity of the algorithm does not depends of the value of k
- Dependant of the structure used to find neighborhoods [KD-Tree or Ball-Tree]
- Less effective on small datasets than K-means

Reminder : Louvain is a "2 steps" algorithm



Architecture of Automeans



Large analysis (Automeans)

Iterations

Fine analysis (Algo **B**)

Final result

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. **A short conclusion**
12. Future work : next steps

A short conclusion

- **Scalability** : it's ok. We are able to clusterize a hundred of million of malware (with a larger RAM !)
- **Quality** of clusterization : well, it depends which sub FV (components of the full FV) you have chosen
- We think it is an interesting tool to **understand** your $\mathcal{Blob}$
- And of course : Challenge is still open
- A lot of work can be done to have a better tool
- "Long" paper : ASAP.

Contents

1. Authors & Acknowledgment
2. Our problem : Malwarology
3. Strategies & Tactics
4. Some problems to solve
5. Machine Learning in a Nutshell
6. Some (minor) problems to solve
7. Architecture
8. Our experiments : Strategy & Tactics
9. Some results
10. Automeans
11. A short conclusion
12. **Future work : next steps**

Future work

- Automeans : a new algorithm, faster than Kmeans on our $\mathcal{B}lob$ (it computes automatically the best k, done, published asap)
- From Packed to unpacked malware : use an *unpacker* (done, we have one) and use the full double FVs
- Deep Learning : autoencoders, siamese, LSTM etc. (in progress, promising)
- Graphs : t-spanner graphs (in progress)
- From static analysis (parser) to dynamic analysis (sandbox) : how to combine them ? (in progress, we have a dedicated sandbox)
- Towards an *automatic generation of Yara Rules* : (complex but look [ref 3] with [ref 2])

Authors & Acknowledgment

Our problem : Malwarology

Strategies & Tactics

Some problems to solve

Machine Learning in a Nutshell

Some (minor) problems to solve

Architecture

Our experiments : Strategy & Tactics

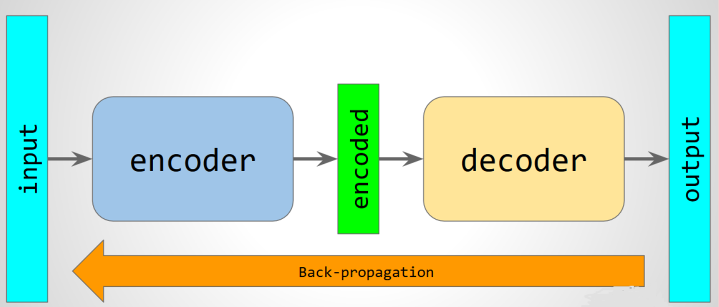
Some results

Automeans

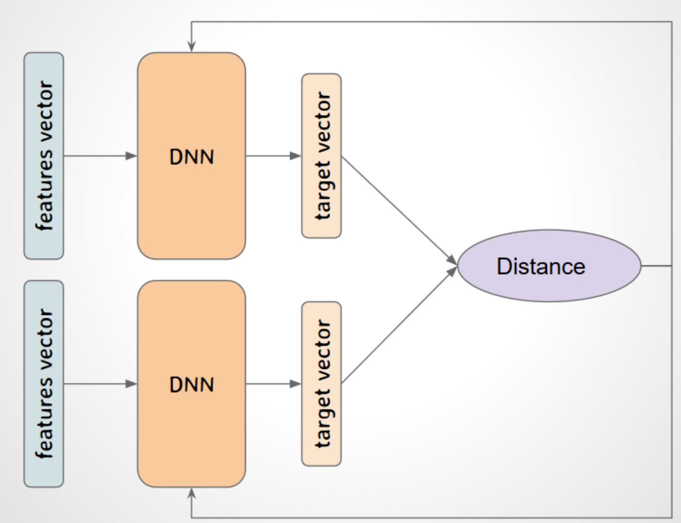
A short conclusion

Deep Learning : Autoencoder

Use an *Autoencoder* to construct new features vectors with smaller dimension



Deep Learning : Siamese Net or how to learn a distance



Some references

1. Ember Malware Dataset :
   https ://arxiv.org/pdf/1804.04637.pdf
2. theZoo Malware Dataset : https ://github.com/ytisf/theZoo
3. A Python Notebook (by S. Larinier aka @Sebdraven) : Python and Machine Learning : How to clusterize a malware dataset ?
   https ://github.com/sebdraven/hack_lu_2017
4. PeToJson : https ://github.com/sebdraven/petojson
5. *Malware Data Science Attack Detection and Attribution* : by Joshua Saxe with Hillary Sanders :
   https ://nostarch.com/malwaredatascience
6. MISC HS N. 18 (in French) : "Machine Learning et Sécurité" : https ://boutique.ed-diamond.com/en-kiosque/1363-misc-hs-18.html