Python and machine learning:

How to clusterize a malware dataset ?

Agenda

- Setting up your environment
- Numpy for the win
- Machine Learning: definitions and generalities
- How to make a good features vector with malware ?
- Algorithms: K-Means and DBScan
- Application on the dataset the Zoo

Little presentation

- Sebastien Larinier (CEO of SCTIF)
- @sebdraven
- https://github.com/sebdraven
- Pythonist
- DFIR, malware analysis
- Honeynet project chapter France and co organizer of Botconf

Thanks a lot

- Alexandre Letois and Robert Erra for training me at machine learning
- Paul Rascagneres for answering my many questions about PE format
- Alexandra Toussaint for trying to correct my bad english and listening to me everydays for one year when I was speaking about malware classification and machine learning
- DGA, Epita and Sekoia for “Rapide” project ViralStudio

Setup and activate your environment

Instructions are described here to setup your environment and activate it

Numpy and Scipy for the win

Click on the notebook: Numpy and Scipy for the win

Machine Learning: Definitions and generalities

Feature: characteristics of the object useful for algorithms

Vector of features: an array of features

Cluster: a group of objects decided by an algorithm

Label: name of the cluster

Machine Learning: Definitions and generalities

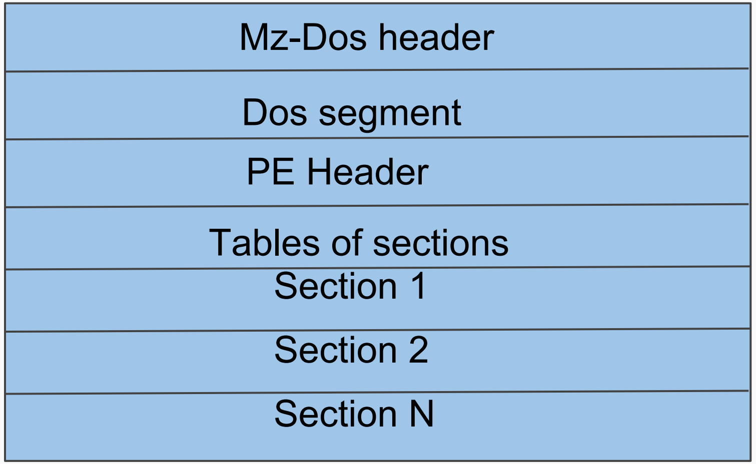
Algorithm supervised: the system is trained on a labeled dataset and we know the number of clusters. The new elements are classified in a existent cluster.

Algorithm unsupervised: the system is trained on an unlabeled dataset and we don't know the number of clusters. The clusterization must be verified before classifying new elements

We used as an input a matrix where each row is a vector of features and each column is a feature

PE format: definitions and generalities

Pe format


Mz-Dos header
Dos segment
PE Header
Tables of sections
Section 1
Section 2
Section N

Pe format

```
typedef struct _IMAGE_DOS_HEADER
{
    WORD e_magic;               //magic number
    WORD e_cblp;                //Bytes on last page of file
    WORD e_cp;          //Pages on file
    WORD e_crlc;        //relocations
    WORD e_cparhdr;     // Size of header in paragraphs
    WORD e_minalloc;    // Min extra paragraphs needed
    WORD e_maxalloc;    // Max extra paragraphs needed
    WORD e_ss;          // Max extra paragraphs needed
    WORD e_sp;          // Initial SP value
    WORD e_csum;                // Checksum
    WORD e_ip;          // Initial IP value
    WORD e_cs;          // Initial (relative) CS value
    WORD e_lfarlc;              // File add of relocation table
    WORD e_ovno;                // Overlay number
    WORD e_res[4];              // Reserved words
    WORD e_oemid;               // OEM identifier
    WORD e_oeminfo;     // OEM information
    WORD e_res2[10];    // Reserved words
    LONG e_lfanew;              // File addr of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Pe Format

- PE header

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                   Signature;
  IMAGE_FILE_HEADER       FileHeader;
  IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

- PE\0\0

Pe format

- PE header

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

Pe format

- PE header

```
Typedef struct _IMAGE_OPTIONAL_HEADER {
 WORD              Magic;
 BYTE             MajorLinkerVersion;
 BYTE             MinorLinkerVersion;
 DWORD              SizeOfCode;
 DWORD              SizeOfInitializedData;
 DWORD              SizeOfUninitializedData;
 DWORD              AddressOfEntryPoint;
 DWORD              BaseOfCode;
 DWORD              BaseOfData;
 DWORD              ImageBase;
 DWORD              SectionAlignment;
 DWORD              FileAlignment;
```

Pe format

```
WORD        MajorOperatingSystemVersion;
  WORD        MinorOperatingSystemVersion;
  WORD        MajorImageVersion;
  WORD        MinorImageVersion;
  WORD        MajorSubsystemVersion;
  WORD        MinorSubsystemVersion;
  DWORD       Win32VersionValue;
  DWORD       SizeOfImage;
  DWORD       SizeOfHeaders;
  DWORD       CheckSum;
  WORD        Subsystem;
  WORD        DllCharacteristics;
```

Pe format

```
DWORD         SizeOfStackReserve;
  DWORD         SizeOfStackCommit;
  DWORD         SizeOfHeapReserve;
  DWORD         SizeOfHeapCommit;
  DWORD         LoaderFlags;
  DWORD         NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Pe Format

- Pe Header

```
typedef struct _IMAGE_DATA_DIRECTORY {
DWORD VirtualAddress;
DWORD Size;
} IMAGE_DATA_DIRECTORY,*PIMAGE_DATA_DIRECTORY;
```

Pe format

Position	Name	Description
0	IMAGE_DIRECTORY_ENTRY_EXPORT	Export table
1	IMAGE_DIRECTORY_ENTRY_IMPORT	Import table
2	IMAGE_DIRECTORY_ENTRY_RESOURCE	Ressources table
3	IMAGE_DIRECTORY_ENTRY_EXCEPTION	Exceptions table
4	IMAGE_DIRECTORY_ENTRY_SECURITY	Certificats table
5	IMAGE_DIRECTORY_ENTRY_BASERELOC	Relocalisations table
6	IMAGE_DIRECTORY_ENTRY_DEBUG	debug
7	IMAGE_DIRECTORY_ENTRY_COPYRIGHT / IMAGE_DIRECTORY_ENTRY_ARCHITECTURE	copyright
8	IMAGE_DIRECTORY_ENTRY_GLOBALPTR	Global pointer
9	IMAGE_DIRECTORY_ENTRY_TLS	Threads table
10	IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	Configuration table
11	IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT	Bound import
12	IMAGE_DIRECTORY_ENTRY_IAT	...
13	IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT	...
14	IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR	...
15	-	vide

Pe format

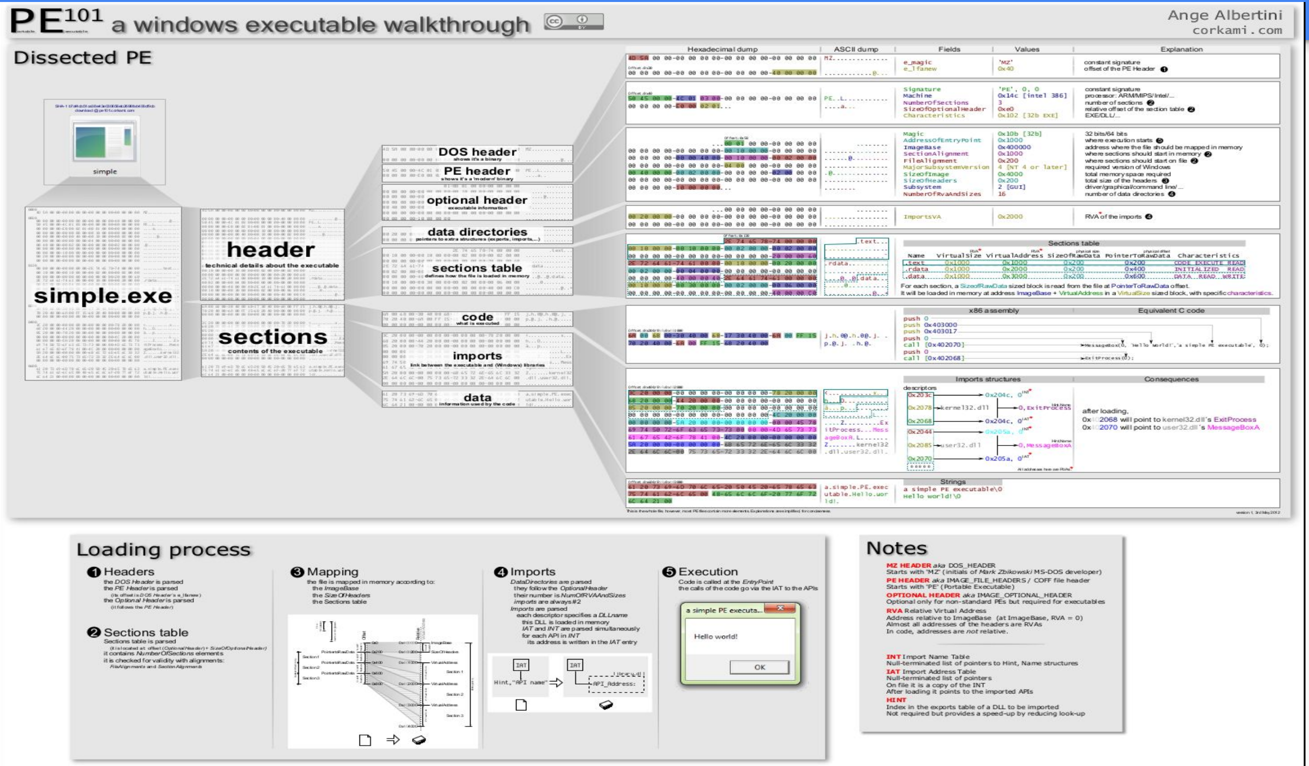
- Table of sections

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Pe Format

- Imports table
  - Functions used by the binary from external lib
- Exports table
  - Functions shared by the binary (basically DLLs)
- Ressources Tables
  - icons, strings, langage using

Pe Format



PE101 a windows executable walkthrough
Ange Albertini
corkami.com
Dissected PE
simple.exe
header
sections
DOS header
PE header
optional header
data directories
sections table
code
imports
data
Loading process
Headers
Sections table
Mapping
Imports
Execution
Notes

PE and Featuring

How to transform a PE into an array to make a vector of features

First step is to extract data in json files.

Open the notebook PE and Featuring

Malwares and featuring

Interesting informations for a malware:

1. Sections: name,size, entropy, characteristics
2. Imports: number of modules, number of symbols, functionalities
3. Exports: number of modules, number of symbols, functionalities
4. Size of file

First Vector of features

So we make a first feature vector:

[size of file, number of sections, median of entropy, number of imports, number of exports]

So we record all vectors in redis table

Test of the first vector

Test the first vector of features with Kmeans and DBscan Algorithms to check if our vector is correctly designed

K Means Algorithm

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster “centroids”; note that they are not, in general, points from X, although they live in the same space. The K-means algorithm aims to choose centroids that minimise the inertia, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_j - \mu_i||^2)$$

K Means Algorithm

First step:

- We choose the number of cluster
- We choose the initial centroids in three ways:
  - With k-mean ++ init, the algorithm chooses k centroids in processing an index called inertia in a loop and choose the better value
  - Randomly, the algorithm chooses k centroids in the matrix
  - Nparray, the user chooses the k centroids

K Means Algorithm

Second step:

- The algorithm calculates distance between k- centroids and all vectors in the matrix and constructs k clusters minimizing inertia with k centroids and the nearest vectors with this k centroid

Results with the first vector

The first results are interesting but it's not totally efficient.

If you're check the norm of vectors, the size of file is the feature which erase all other values

So we normalize with the max value of each features

Second vector of features

Now we normalize the vector of features:

[size of file / max(size of all files), number of sections/ max(number of sections of all files), median of entropy /max(median of entropy of all files), number of imports / max(number of imports of all files), number of exports / max(number of exports of all files)]

Results with second vectors

The classification is better than the first one, whereas we just normalized the values.

Now we add check with DBscan algorithm with the first and second vectors

DBScan Algorithm

The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, min_samples and eps, which define formally what we mean when we say dense. Higher min_samples or lower eps indicate higher density necessary to form a cluster.

Results with the first vector

It’s worse than K-Means because the vector is depending on the size of file and this make the density bad. We have to normalize the vector.

Results with the second vector

We have a good classification by families and by versions of families

Conclusions

We have seen machine learning is not magic, a work of featuring must be done including the of the dataset.

Here, our dataset is very heterogeneous with a big cluster of EquationGroup, and others clusters with few malwares

The machine learning is useful to make a first filter to clusterize a big dataset because the algorithms have been thought to be scalable contrary to algorithms which compare signatures. (ssdeep,impfuzzy,machoc…)

Yara rules Generation

Yaragenerator

- https://github.com/Xen0ph0n/YaraGenerator
- Generate automatically yara rules based on an intersection of strings

Using ours results of clustering malware

On the EquationGroup Cluster we have a rule matching this family.

But if we try with the Regin family, it doesn't work because the tool doesn't find an intersection based on strings.

Conclusions

Conclusions

Machine Learning is not magic.

It can help for filtering data and it must be mix with others techniques to be useful.

Thanks

Thanks for your attention !

If you have questions, don’t hesitate !

slarinier@gmail.com

@sebdraven