A Quick Summary of Using machine learning to analyze malware

Peiju Li

May 26, 2018

Introduction

Malware detect is constantly a main focus in the security area. With the rapid development of internet, cyber security being more and more important than before. The digital fraud and malware also highly increased. As a new and popular method, machine learning brings vitality to traditional virus analysis. The goal for this awesome project is to develop a malware classifier with machine learning method and gain more knowledge about security and machine learning while self-researching.

Background

Different malware types

Virus, Worm:

This type of malware are just a software that after you loaded, install, it will spread out and infect other benign software.

Trojan:

Hiding the malware inside the benign software, and letpeople think are good and download it. It can be employed by cyber-thieves and hackers trying to gain access to users' system.

Rootkit:

The rootkit is designed to provide privileged access to a computer. normally, it requires the admin control, it usually hiding inside the system, and with invisible commands, it's pretty hard to detect them.

Ransomware:

The ransomware encrypt the machine data, freeze all the operations of victim, and ask victim to transfer the money otherwise just damage those files.

Keylogger:

log the key that you entered, for example, your password, your bank account, and latter use those information to attack.

Static and dynamic analyze

There are two main methods of analyzing malware, static method and dynamic. Static method is mainly analyzing the files. Every software need source code, so the static method is basically scanning those binary files, and extract using various techniques such as file format inspection, and disassembly.
Static method has these benefits:
can be performed pre-execution;

work on invalid executables;
computationally inexpensive;
Compared to dynamic analysis, the signature-based analysis is a static method that relies on predefined signatures. Fingerprint such as MD5 OR SHA1 hashes, and by detecting the hash value of these signatures, it is detected by that anti-virus.
I am interested in dynamic approach as well, and it includes the APIs, the features which are suspicious including the registry key change,“ connection established”, “permission changed”, etc. With checking how many features that one is triggered, and added different weight to each parameter, machine learns approach is being used. This is a very interactive way and no need to preload the dataset, but this is also a bit riskier if not operate suitably. The method of using signature based is an old technology, the signatures are created by examining the internal components of an object and malware authors simply modify these components while preserving the object's functionality and behavior, result in the malware can alter its signature to avoid detection. The method could be used to change the register name, shrinking code, and insert to garbage code etc.

File format inspection

Portable executable Files could contains huge informations, as it contains .
Although I know this feature has limited to Windows system, there are a large number of users using Windows system that may be infected by Malware by .exe, .ddl and so on. The analysis on PE File could help on these users.

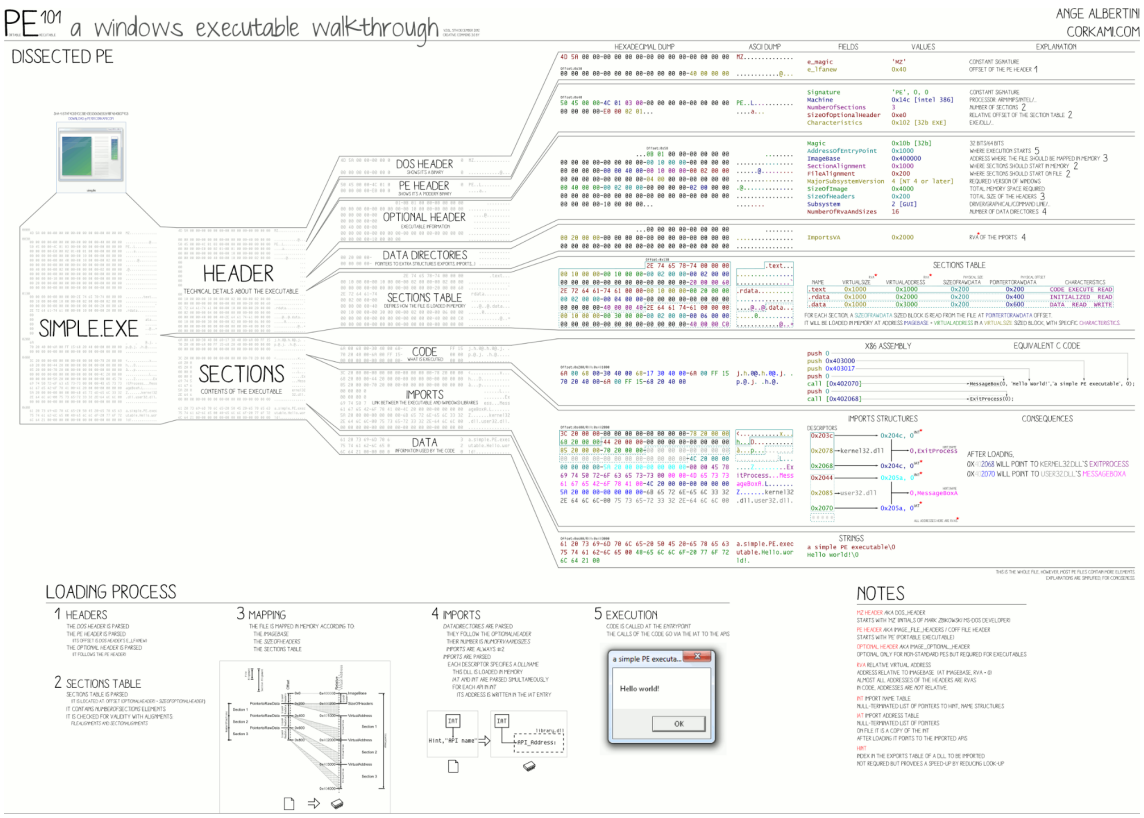


Figure 1: source

The Portable Executable (PE) format is a file format for executables, object code, DLLs, FON Font files, and others used in 32-bit and 64-bit versions of Windows operating systems. The PE format is a data structure that encapsulates the information necessary for the Windows OS loader to manage the wrapped executable code.
When a PE file being executive, the PE loader inspect the movements inside the DOS MZ header. If found, then jump to there. PE loader inspect if PE header is suitable, following is the section table, using file image, image them into the sections.

then import table logic parts. Execute at the end of this process. This is an .exe file inside structure.
The layout of PE File:
A PE file consists of a number of headers and sections that tell the dynamic linker how to map the file into memory. An executable image consists of several different regions, each of which requires different memory protection; so the start of each section must be aligned to a page boundary. For instance, typically the .text section (which holds program code) is mapped as executing/read-only, and the .data section (holding global variables) is mapped as no-execute/read-write.

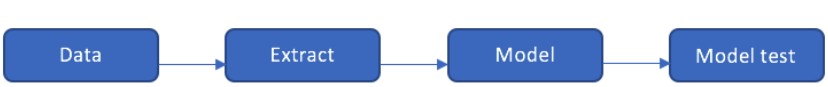
Disassembly

This method is reversing the machine code to assembly language and inferring the software logic and intentions, and utilize the analyze of feature extracted train the model.
I looked into Kaggle Malware detect competition first place paper, and then do research on the method of NGram. They analysis the 1 gram, 2 gram, 4 gram, calculate the percentage of a full line gram weight and pick the 40,000 most popular presented patterns from the data were given.
The n-gram is a subset of a word. Techniques of this method are first used in detecting the spelling errors in a words. In here, we know that the source code has some same structures, by looking at some special character, notice that aways it contains some fixed structure each Section start with a colon and point" : " SECTION.HEADER.image .SizeOfCode etc.
The first place team (Reference) extracted features from following aspects.
On file properties: size, compressed size and ratios between them. On contents of the asm files: sections, dlls, opcodes and interpunction statistics by section. On contents of the bytes files: 1, 2,and 4 grams, full lines and distribution of entropy.

Different machine learning methods

Machine learning is a tool that could used to mining the data features with statistical model, neural network models. The basic idea of supervised machine learning is using dataset or small sample to train a model, and then based on different algorithm to perform the classification, regression, and so on.

$$y_{pred} = X y_{input}$$



Decision Tree method

Decision trees are the data structures that have a tree-liked graph. You can define the level of depth you want to goto, each decision is T or F, with a certain probability.

Extra tree classifier

It's based on a number of randomized decision trees. We can define some criterion, the maximum features, the tree depth for better tuning of the model.

Random Forest method

It is an ensemble learning method of classification and regression. The algorithm for random decision forests was using the random subspace method.

How to build the tree

As we can see, there are three method are using the decision tree, so I looked into how they build up the tree. -let N represent the number of training set data, M represents the number of character; -from N samples, take N times, and form a training set, use the unused sample as prediction and

judge the error;
-To each node, randomly select m characters, and decide each node on these characters, calculated best punching;
-Each tree will grow up with their own forms.
Well, this is the theoretical of decision tree, but with using sklearn library, it can be done, with extra tunning and modification.

Data process

With trying different method of downloading Kaggle Malware detect dataset and failed, as I haven't found out a kernel with direct linked to this database, and the training set and testing set is too big(the test size is 17.6G, and train folder is 17.5G). I tried to handle those input at cloud base, but then I failed after overnight the network is not keep live. I tried to connect the dataset remotely to the cloud, unluckily I haven't success. So rather than using the resource of this competition and processing the asm files, I decided to look further in File format inspection. To find the smaller dataset, VirusShare is mentioned in some papers, but when I visit it want to find those files, I found it ask for permission. Although I felt a bit disappointed, I think it pretty good design. If anyone, e.g. me can access, they cannot tell if this user wants to learn from the virus or use them to do some illegal activities. While but there is a list of citing, so I got two pefiles for the purpose of practicing myself how to extract raw feature from files and I output those features into a csv, latter I found out the dataset from urwithajit9/ClaMP with extracted feature in csv, so I run the model on this dataset and append the other very small set of 89 files extracted features on it.
The library of extracting from binary files a file infraction is pefile in python. Then with simple calling about the sections of pefile. for example, pe.OptionalHeader.SizeOfImage will give you the size of image of this file.

Problem define

We are aim to output this file is good or malicious, given a file and extract the characteristic and then predict it. the square loss function is both convex and smooth and matches the 0–1 indicator function when $yf(\vec{x}) = 0$ and when $yf(\vec{x}) = 1$. However, the square loss function tends to penalize outliers excessively, which result to converge slower.

Evaluation metric

As the data collected are weighted equally for good files and malicious files, so no need to take the *logloss*, just use the traditional

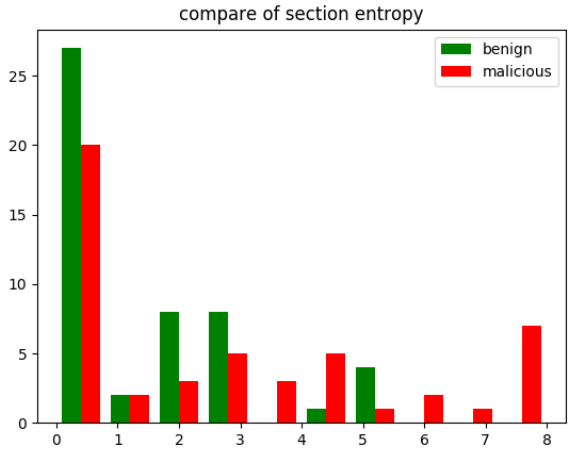
Handle exception

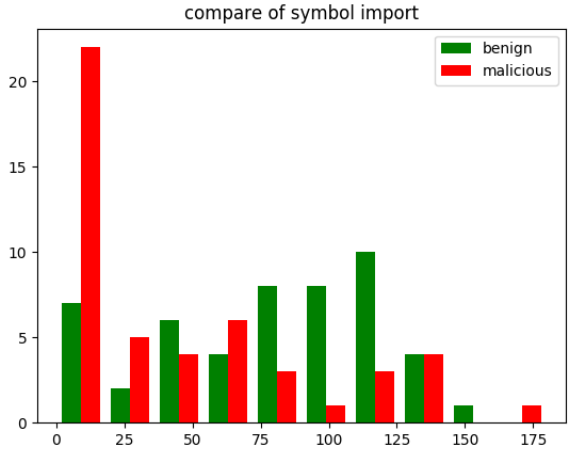
As not every file have all the features that we want to extract, so the features that nearly all files maintain will be put into dense features, the features that not appears very often will be put into sparse features.
Another type of exception is the type error. As the machine learning basically is just matrix multiplication, and it only accept integer and double, so for the string type, we need to relabel it categorically. For example, the version number, they are strings. and for image directory, we can either name,
0:'image dir export',
1:'image dir import', 2:'image dir extr resource' or we can make three such column and represent each in a column.

Modeling

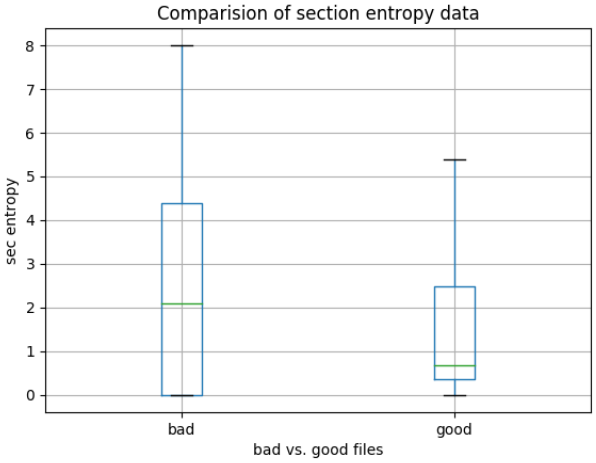
So after extract features from raw files in data procession section, I have already got 199 features for each binary pe file, and if added up to the label of benign or malicious, there are 200 features in total.(For another dataset, there are 56 features). So how to determine which features to use? How to make it can be selected out next time in machine learning?

We can have some guess and plot the features we guess. I plot the section entropy and import symbols, cause the entropy normally could reflection how many information we have.



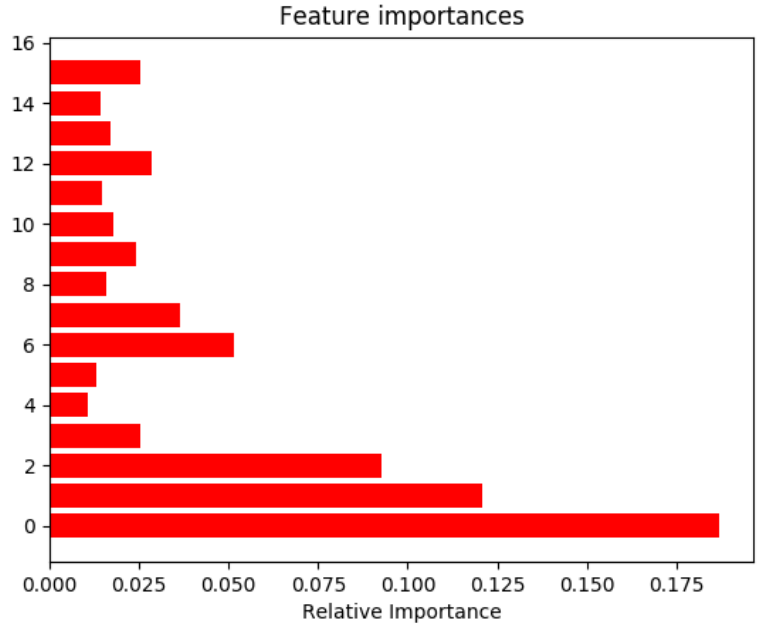


Well, but from the box plot I did, although the mean of bad and good has notable difference, but the variance is quite big, the bad files range containing good files, so it is not a very good indicator.(The machine learning method feature selected also showed that, it ranked around 14th for the dataset)

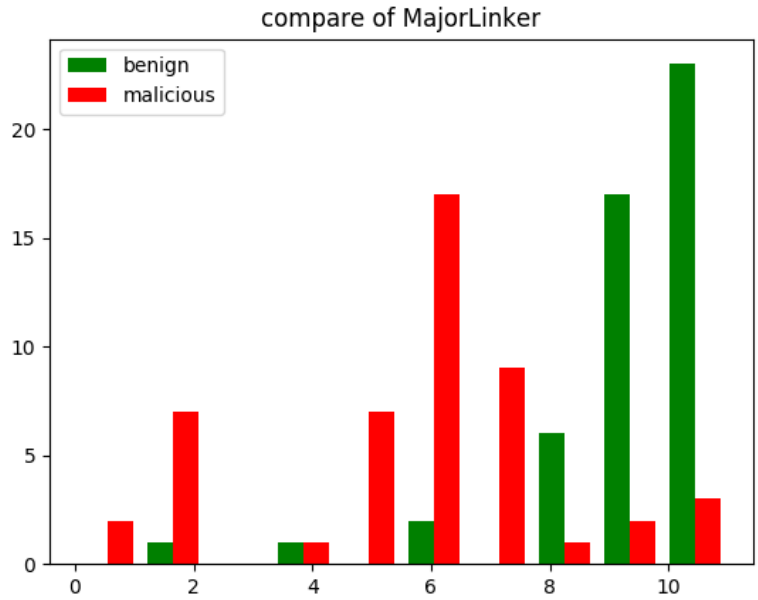


Feature importance

Well, but to plot one by one is pretty painful, so I used the sklearn library, sklearn.feature Selection. Computed the top important features. For the purpose of comparing the algorithms of select features, I compute them with different algorithm and plot them to find the best accuracy. Then among 'variance threshod', 'l1 linear SVC', and 'extra trees classifier', the extra tree perform best.



After features importance, I found that the Major linker and minor linker is also an important feature in both dataset. Then I do some research in it what is the linker. The linker worked with diverse programming languages, while the compiler was language specific. However, the .Net world has merged the next two bytes of the Image Optional Header to contain the version number. Earlier, the major version and the minor version of the product that created this file were stored. This number normally denotes the version of Visual Studio that created this file. But it has an obvious shorthand, as the windows system can change version quite often, and the malware has the trend of virus rejuvenation, so it might not be a good prediction of future emerging malware files, but it still a good indicator to the old virus.




```
<For the small set>

1. feature generated_check_sum (0.099450)
2. feature sec_rawptr_rlpack (0.056860)
3. feature pe_warnings (0.047196)
4. feature pe_minorlink (0.040621)
5. feature number_of_bound_import_symbols (0.038537)
6. feature datadir_IMAGE_DIRECTORY_ENTRY_BASERELOC_size (0.037478)
7. feature pe_char (0.032197)
8. feature datadir_IMAGE_DIRECTORY_ENTRY_EXPORT_size (0.032055)
9. feature MajorLinkerVersion (0.030821)

<For the large set>
1. feature MajorLinkerVersion (0.234059)
2. feature SizeOfStackCommit (0.113248)
3. feature FileAlignment (0.110251)
4. feature SizeOfStackReserve (0.041102)
5. feature SizeOfHeapReserve (0.039304)
6. feature SizeOfImage (0.036561)
7. feature MajorImageVersion (0.030623)
8. feature NumberOfSymbols (0.030120)
9. feature PointerToSymbolTable (0.028437)
10. feature NumberOfSections (0.027913)
11. feature MinorOperatingSystemVersion (0.026544)
12. feature SizeOfInitializedData (0.026437)
13. feature SizeOfHeaders (0.024357)
14. feature DllCharacteristics (0.022569)
15. feature BaseOfData (0.020538)
16. feature CheckSum (0.019730)
```

The intersting thing happens that each time run the tree classifier, there will be different features ranks each time. Suppose we want to divide to two classes, then the Tree Classifier works like repetitively divides the working area(plot) into sub part by identifying lines. With large sample input, the tree classifier works reasonably good.

Model performance

I am comparing among these methods, Extra tree, XG boost, gradient boost, Random Forest, Decision tree. With analyzing type I(false positive) and type II error(false negative), we aim for lower both type of error, but to minimize the type II error is more important, cause if we missed a malware, it still could result to disaster. I think all of them works pretty good, but the extra tree classification of feature extract with random tree performs best, with 2.6% type 1 error and 1.0% type 2 error.

Feature Engineering	Training Model	Accuraccy (4 s.f.)
Raw features without data pre-processing	XGBoost (without parameter tuning)	96.3462 type1 error: 4.08% type2 error: 1.27%
Important list of features with extra tree	XGBoost	97.6856 type1 error: 4.01% type2 error: 1.21%
Important list of features with extra tree	Gradient Boosting	97.3964 type1 error: 4.49% type2 error: 1.64%
Important list of features with extra tree	Random Forest	98.2077 type1 error: 2.65% type2 error: 1.09%
Important list of features with extra tree	Decision Tree	96.6248 type1 error: 6.14% type2 error: 0.91%
Important list of features with extra tree	Cross-validation	(Too long to train) 95.0341
Important extracted by with random tree	XGBoost (without parameter tuning)	97.6910 type1 error: 3.88% type2 error: 1.11%
Important extracted by with random tree	XGBoost	97.7113 type1 error: 3.79% type2 error: 0.91%
Important extracted by with random tree	Gradient Boosting	97.3256 type1 error: 3.88% type2 error: 1.45%
Important extracted by with random tree	Random Forest	97.8748 type1 error: 3.06% type2 error: 1.27%
Important extracted by with random tree	Decision Tree	96.5284 type1 error: 6.13% type2 error: 1.09%

Conclusion

Although the accuracy is pretty high, but we still cannot say it good enough. The thousands thousands source code need to analyze in a very short time, and if 1.0% type 2 error, that means,on average, in 100 files, there will be missing 1 virus.
But throughout the semester, I start from 0 knowledge of security and machine learning to now, finished up the project, I learned the different malware types, different analyze methods, and the file structures, although I haven't used some of them in this project, nevertheless, it gives me a more broader view about malware analyze except for limiting to the static file format. There are still a lot of thing I can explore.