Machine learning approach for malware multiclass classification

Houssem Hosni

Machine learning approach for malware multiclass classification

Houssem Hosni
LIP6 - Sorbonne University
houssem.hosni@lip6.fr

***Abstract*— Categorization of modern malware samples on the basis of their behavior has become essential for the computer security community, because they receive huge number of mutated malwares every day, and the signature extraction process is usually based on malicious parts characterizing malware families. Microsoft provided the data science and cybersecurity community with an unprecedented malware dataset of near 0.5 terabytes, containing more than 20K malware samples to encourage open-source progress on effective techniques for grouping variants of malware files into their respective families. In the present paper we develop an effective machine learning approach where emphasis has been given to the phases related to data analysis, feature engineering and modeling. The proposed methodology gave interesting classification results in terms of adopted performance metrics.**

Keywords*: *Windows Malware, Machine learning, Malware family, Computer security, Classification, Feature engineering, Accuracy, Performance metrics, KNN, Xgboost, random forest...

I. INTRODUCTION

In recent years, Well funded and multi-player syndicates heavily invest in technologies and capabilities built to evade traditional anti-malwares protection, requiring vendors to develop counter-mechanisms for detecting, categorizing and deactivating them. In the meantime, they inflict significant financial loss to users of computer systems. One of the major challenges that anti-malware software faces today are the vast amounts of data which needs to be evaluated for potential malicious intent. One of the main reasons for these high volumes of different files is that in order to evade detection, malware authors introduce mutation to the malicious components, namely polymorphism and metamorphism. This means that malicious files belonging to the same malware family, with the same forms of malicious behavior, are constantly modified and/or obfuscated using various tactics, so that they appear to be different files. A first step in effectively analyzing and classifying such a large number of files is to group them and identify their respective families.

II. PROBLEM STATEMENT AND OBJECTIVES

The term malware is a contraction of malicious software. Put simply, malware is any piece of software that was written with the intent of doing harm to data, devices or to people.[1] The major part of protecting a computer system from a malware attack is to identify whether a given piece of file/software is a malware and identify its family class. Malware detection and classification techniques are two separate tasks which are performed by anti-malware and cybersecurity companies[2]. Once detected, malware needs to be categorized into a specific family for further analysis. The aim behind this process is to improve anti-malware systems reactivity and performance. As far as a very high number of malware variants is concerned, the need for the automation of this process is clear-cut. In this work we discuss a novel machine learning approach that reaches a very interesting classification accuracy of nearly 99.5%.To do so there are some objectives and constraints that we have to deal with.(1) Converting malware files into simple data points(featurization).(2) The error of this probabilistic classification process has to be minimal.(3) Minimizing latency as malware classification should not take a long time.

III. MACHINE LEARNING PROBLEM

A. Data overview

In this section we try to understand the data that has been provided. The original data collection was published by Microsoft on the Kaggle competition website[3]. The dataset contains 10,868 samples of *.byte files and 10,868 of *.asm files. It consists of roughly 200GB data out of which 50Gb of data are *.bytes files and 150GB of data are *.asm files. To each malware corresponds a *.byte file with hexadecimal representation of binary content and an *.asm file from assembly view. The real name of each malware is replaced by a unique 20-character hash value which we call ID. And the portable executable (PE) header is also removed to ensure sterility. The PE header describes the rest of the file. Basic information in PE header of a Windows PE file includes a DOS header and related data, NT header, a section table and section data. It provides rich attributes of the PE file[4]. The malwares mainly belong to nine different families: Ramnit, Lollipop, Kelihos-ver3, Vundo, Simda, Tracur, Kelihos-ver1, Obfuscator.ACY, and Gatak. The data collection provides hex code and disassembled code formats. Figure 1 and Figure 2 are taken from the same malware. Figure 1 is from the hexadecimal(binary) view while Figure 2 is from the assembly view.

```
00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
00401010 BB 42 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 08 BB 42 00 E9 26 1C 00 00 CC CC CC CC CC
00401030 56 8B F1 C7 06 08 BB 42 00 E8 13 1C 00 00 F6 44
00401040 24 08 01 74 09 56 E8 6C 1E 00 00 83 C4 04 8B C6
00401050 5E C2 04 00 CC CC CC CC CC CC CC CC CC CC CC CC
00401060 8B 44 24 08 8A 08 8B 54 24 04 88 0A C3 CC CC CC
00401070 8B 44 24 04 8D 50 01 8A 08 40 84 C9 75 F9 2B C2
00401080 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
```

Fig. 1. Part of Hex code

Disassembled file is generated from the corresponding hex code by the Interactive Disassembler (IDA) tool[5]. As shown in figure 2, disassembled files translate binary machine code into assembly language which is meaningful for

humans. For example, hex decimal digits 56 is disassembled as instruction push and the object register esi.

```
.text:00401000                                assume es:nothing, ss:nothing, ds:_data,
   fs:nothing, gs:nothing
.text:00401000 56                          push    esi
.text:00401001 8D 44 24    08                      lea     eax, [esp+8]
.text:00401005 50                          push    eax
.text:00401006 8B F1                          mov     esi, ecx
.text:00401008 E8 1C 1B    00 00                      call    ??0exception@std@@QAE@A
```

Fig. 2. Part of Disassembled Code

B. Mapping to a machine learning problem

In this section we convert the problem stated above to a multiclass classification problem.

Performance Metrics: The models are evaluated using logarithmic loss and confusion matrix(accuracy). For each file, we generate a set of predicted probabilities (one for every class) and the logloss of the model is as follows:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} log(p_{ij}) \quad (1)$$

where N is the number of files in the test set, M is the number of labels, log is the natural logarithm, y_{ij} is 1 if observation i is in class j and 0 otherwise, and p_{ij} is the predicted probability that observation i belongs to class j. The way we split our dataset will be as follows: 80% for train and cross-validation(64% - 16%) and 20% for test.

IV. Exploratory Data Analysis And Feature Engineering

A. Working on *.byte files

The first task is to separate *.byte files from *.asm files as we will do all the modeling and the data analysis on those files separately. To verify if our dataset in balanced it is important to know the frequency of each class as it is shown on the following histogram.

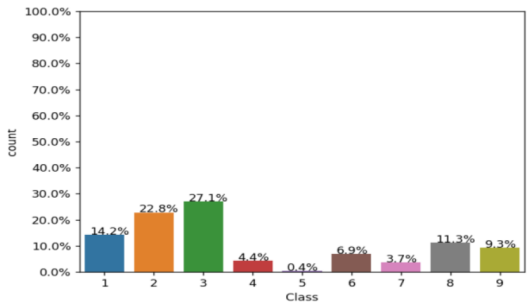


Fig. 3. distribution of malware classes across *.byte data

Next task is feature engineering on *.byte files. The first feature to be considered is the size of the file. One way to know if the size is a useful feature/variable, is the following boxplot which displays the variation of file sizes per classes. As it can be observed, file size has some usefulness that can help us to detect and differentiate between some classes and consequently would help us predict malware classes.

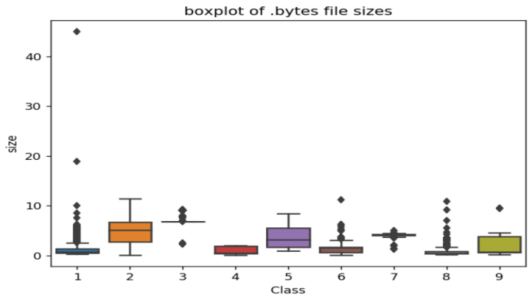


Fig. 4. *.byte file size per malware class

The *.byte files consist of hexadecimal numbers (256 decimals from 00 to FF). We consider this as a text problem and we build our bag of words(unigrams) which represent the occurrence/frequency of those symbols in each file. We just add the malware class and its *.byte files size as additional features/variables. The next step is to normalize and standardize all the variables. The first five rows of the resulting dataset are shown on following figure:

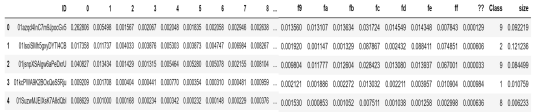


Fig. 5. Resulting dataset (*.byte)

B. Working on *.asm files

There are 10868 *.asm files and they make up about 150 Go. And there are almost 6 main components that well define an *.asm file and they are as follows: Address, Segments, Opcodes, Registers, function calls, APIs To process all this amount of data we use multithreading and parallel computing. We split the 150 Go into 5 folders of 30 Go. With the help of parallel computing we extract all the features plus the size of each *.asm file. The 52 features to be considered:
***Prefixes*:**['Header:','.text:','.Pav:','.idata:','.data:','.bss:', '.rdata:','.edata:','.rsrc:','.tls:','.reloc:','.BSS:','.CODE'],
***Opcodes*:**['jmp','mov','retf','push','pop','xor','retn','nop', 'sub','inc','dec','add','imul','xchg','or','shr','cmp','call', 'shl','ror','rol','jnb','jz','rtn','lea','movzx'],
Keywords:['.dll','std::',':dword'],
Registers:['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
The following boxplot shows the variation of *.asm file sizes depending per malwares classes.

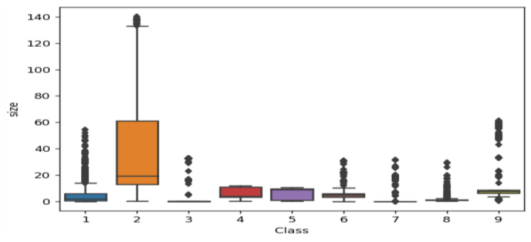


Fig. 6. *.asm file size per malware class

After normalization of data, the resulting dataframe is as follows:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	esp	eip	Class	size
0	01kcPWA9K2BOeQeS5Rju	0.107345	0.001092	0.0	0.000781	0.000023	0.0	0.000084	0.0	0.000072	...	0.000746	0.000301	0.000360	0.001057	0.0	0.026787	0.001468	0.003173	1	0.000432
1	1E93CpP60RHFNTSQhm	0.096045	0.001230	0.0	0.000617	0.000019	0.0	0.000000	0.0	0.000072	...	0.000328	0.000965	0.000686	0.000153	0.0	0.025362	0.000000	0.002198	1	0.000327
2	3eKvw2qZrbTnBcsDfX	0.096045	0.000927	0.0	0.000300	0.000017	0.0	0.000038	0.0	0.000072	...	0.000475	0.000201	0.000560	0.000178	0.0	0.019928	0.000000	0.000985	1	0.000172
3	3X2nYTQaPBiMDrAZLe	0.096045	0.000333	0.0	0.000258	0.000008	0.0	0.000000	0.0	0.000072	...	0.000090	0.000281	0.000059	0.000025	0.0	0.014483	0.000000	0.000657	1	0.000008
4	46OZzdcSKDCFVBY7XWxf	0.096045	0.000590	0.0	0.000353	0.000008	0.0	0.000000	0.0	0.000072	...	0.000112	0.000362	0.000243	0.000084	0.0	0.019928	0.000000	0.001204	1	0.000143

5 rows × 54 columns

Fig. 7. Resulting dataset (*.asm)

V. Machine Learning Models

In this section, we focus on the machine learning modeling part. First, We will briefly present the results obtained by some models on both *.byte and *.asm then we consider with more details Xgboost, the model that gave the best results in terms of performance metrics, namely accuracy and logloss.

A. Different models

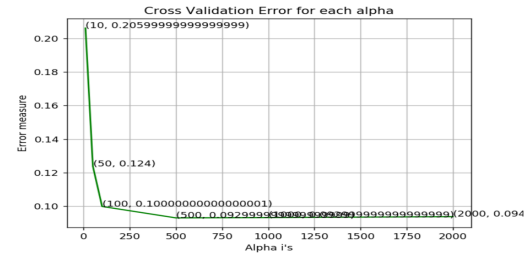
A random model has been used to help us know what's the worst result that a bad model can give. It gave a logloss = 2.48(on test data) and misclassification=88%. For the other machine learning models, we train each one on train data, optimize and tune it's hyperparameter using the cross-validation data and finally we compute it's logloss and accuracy using the test data. The next table shows the results obtained by three different models, namely KNN, logistic regression and random forest that we have trained and used to predict malware classes.

Machine learning model	*.byte files		*.asm files	
	Logloss	Accuracy	Logloss	Accuracy
KNN	0.24	95.50%	0.089	97.98%
Logistic regression	0.528	87.68%	0.415	90.39%
Random forest	0.085	97.98%	0.057	98.85%

Fig. 8. Classification results on *.byte and *.asm separately

B. Xgboost classifier on *.byte files

Xgboost is a great machine learning model[8] that gave us the best results in terms of accuracy and logloss. The hyperparameter alpha we tuned for Xgboost using the cross-validation data was the number of trees. As it's shown on figure 9, the best alpha(that gives the least error) is equal to 500.




```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
```

Fig. 9. Cross-validation error per hyperparameter alpha

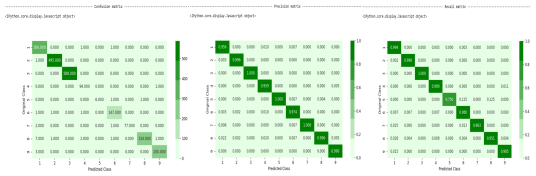
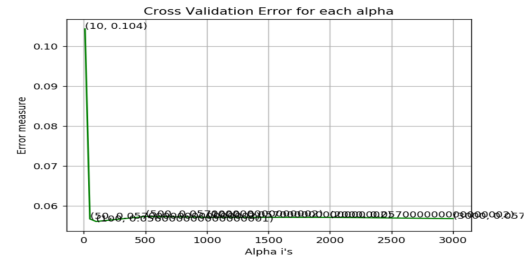


Fig. 10. Confusion/Precision/Recall matrices

Next figure shows the three classification performance matrices related to Xgboost model(on *.byte files).
Of all the data points that have been predicted as belonging to class 5, 100% actually belong to class 5(class 5 was the most difficult class to predict using other models). From the performance and accuracy results shown above its clear that Xgboost performs the best on *.byte files(logloss=0.079,accuracy=98.76%)

C. Xgboost classifier on *.asm files

As we have said above, Xgboost has a lot of hyperparameters that can be tuned but we keep tuning the number of trees: alpha=100 is the best parameter in the case of *.asm files. Xgboost provided the best results on *.asm files as well with a logloss=0.049 and accuracy=99.13%.




```
For values of best alpha =  100 The train log loss is: 0.0117883742574
For values of best alpha =  100 The cross validation log loss is: 0.056075038646
For values of best alpha =  100 The test log loss is: 0.0491647763845
Number of misclassified points  0.873965041398
```

Fig. 11. Cross-validation error per hyperparameter alpha

Next figure shows the three classification performance matrices related to Xgboost model trained using *.asm data.

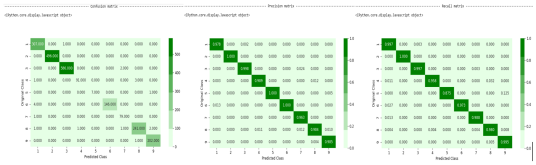


Fig. 12. Confusion/Precision/Recall matrices

VI. Modeling *.asm and *.byte Files Altogether

A. Exploratory data analysis

In this section, we will join the two dataframes obtained using *.asm and *.byte data separately in one global dataframe. This dataframe contains all the hexadecimal and assembly features. By doing so we get the following dataframe(fig 13).

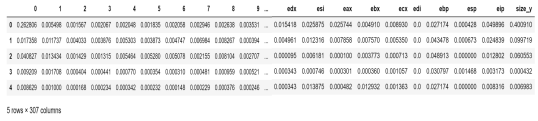


Fig. 13. Resulting dataset (*.byte and *.asm altogether)

Next figure depicts a 2D visualization of our dataframe using a very powerful dimensionality reduction and visualization technique called T-SNE[9]. The visualization was obtained using a perplexity=50.

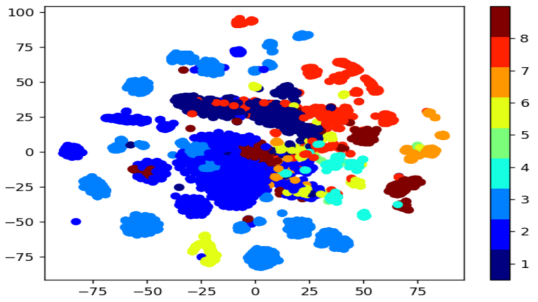


Fig. 14. T-SNE 2D visualization (*.byte and *.asm altogether)

From the figure above we notice that combining/concatenating *.byte features and *.asm features altogether was really a great idea(the clusters of data points that represent the nine classes can be easily identified) and this can let our machine learning models perform and learn much better.

B. Machine learning models

Due to their performance on *.byte *.asm data separately, in this section we focus on random forest and Xgboost.

1) Random Forest: The best alpha (hyperparameter) for random forest is alpha=3000 and it gives a logloss=0.0401 and an accuracy= 99.3% (see fig 15).

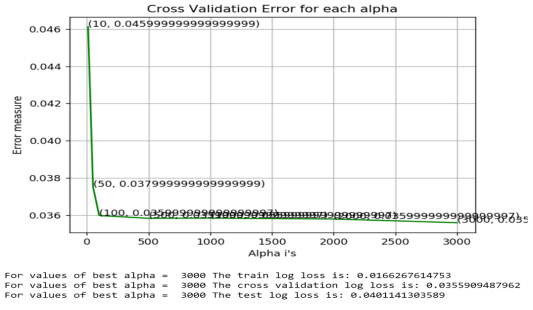


Fig. 15. Cross-validation error per hyperparameter alpha

2) Xgboost: The next figure depicts the hyperparameter tuning for Xgboost. alpha=3000 is the best parameter when mixing *.asm and *.byte files altogether. With a classification accuracy=99.5% and a logloss=0.032, We conclude that Xgboost is the winner between all the models that we tried.

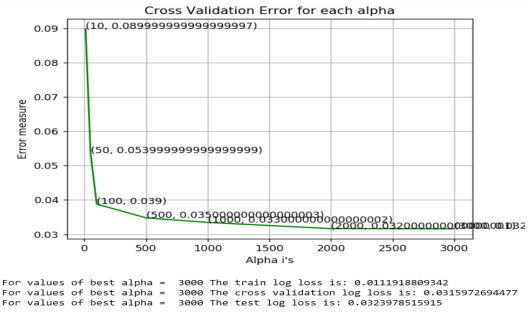


Fig. 16. Cross-validation error per hyperparameter alpha

VII. CONCLUSION AND FUTURE WORK

In this work we presented a malware multiclass classification process from feature engineering to modeling and performance metrics(accuracy,logloss) evaluation. Five learning algorithms were compared statistically for classification performance of malicious files. A dataset has been extracted and normalized from a huge collection of binary and disassembled files representing the content of malwares recorded by Microsoft anti-virus systems. During extraction, 256 hexadecimal features and 52 assembly features were considered. After working on *.byte files and *.asm files separately, a global dataset combining all *.asm and *.byte features was constructed and this task was really useful and brought much more information enabling the machine learning models to learn better and predict malware class much more accurately. For the algorithms and measurements discussed in this work, random forest and XGBoost gave the best results in terms of accuracy score and logloss as performance metrics. As future directions, we aim to extend our approach to consider Bi-grams and Tri-grams and not only simple bag of words. Besides, we will consider using some deep learning models to see if any classification performances improvements can be obtained.

REFERENCES

[1] Malware definition:https://www.avg.com/en/signal/what-is-malware

[2] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto. "Novel feature extraction, selection and fusion for effective malware family classification". In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, CODASPY 16, pages 183194, New York, NY, USA, 2016. ACM.

[3] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, Microsoft Malware Classification Challenge, CoRR, vol. abs/1802.10135, 2018. [Online]. Available: http://arxiv.org/abs/ 1802.10135

[4] Choi, Yang-seo, Ik-kyun Kim, Jin-tae Oh, and Jae-cheol Ryou. 2008. "PE File Header Analysis-based Packed PE File Detection Technique (PHAD)." Computer Science and its Applications, 2008. CSA '08. International Symposium on. IEEE. 28-31. doi:10.1109/CSA.2008.28.

[5] Ida : Disassembler and debugger. https://www.hex-rays.com/products/ida/, 2013.

[6] Lakshmanan Nataraj."Malware Images: Visualization and Automatic Classification". Vision Research Lab University of California, Santa Barbara. URL:https://vizsec.org/files/2011/Nataraj.pdf

[7] W. Aha, Dennis Kibler, Marc K. Albert."Instance-Based Learning Algorithms" (1991).

[8] Xgboost.https://github.com/dmlc/xgboost, 2015.

[9] Laurens van der Maaten, Geoffrey Hinton. "Visualizing Data using t-SNE"(2008)