Malware Classification

by

Kandarp Khandwala and Antara Sargam

A project report submitted to The Johns Hopkins University in conformity with the requirements for the course of Security Analytics

Baltimore, Maryland

March, 2018

Abstract

In the recent few years, we have seen the growth of malware primarily being used to infect computers and systems to extract information or other malicious purposes. A major challenge in the security industry concerning fighting and detecting such malware is the amount of data present. In this project, we use Microsoft's unprecedented malware dataset comprising of thousands of different malware files to showcase our attempt in building a good classification model to predict the class or family a particular malware belongs to based of various features in the dataset.

Thesis Committee

Primary Readers

Professor Lei Ding (Primary Advisor)
Department of Computer Science
Johns Hopkins Information Security Institute

Table of Contents

Chapter 1

Introduction

1.1 Background & Motivation

In recent years, the malware industry has become a well organized market involving large amounts of money. We've seen the rise of well funded syndicates investing heavily in technologies built to evade traditional protection. This requires anti-malware vendors to develop counter mechanisms for finding and deactivating these malware. In the meantime, they inflict real financial and emotional pain to users of computer systems.

The major challenge that security engineers and data scientists face while fighting and detecting these malware is the vast amount of data present around them that needs evaluation. For example, Microsoft's real-time detection anti-malware products are present on over 160M computers worldwide and inspect over 700M computers monthly. This generates tens of millions of daily data points to be analyzed as potential malware.[1]

Another reason this tasks gets complicated is the fact that malware authors introduce polymorphism to the malicious components. This means that the malware which belong to the same "family" might now look different in their behaviour by means of obfuscation and other such tactics. In order to fight this, we first need to be able to group malware in their respective families using Microsoft unprecendented malware dataset involving almost a terrabyte of data.

Therefore, in this project, we try to use our knowledge in data science to come up with the best classification model using multiple classification methods known. We will use this model to best classify different malware into their respective families, based on selecting certain important features from the dataset.

1.2 Understanding the Dataset

The dataset provided by Microsoft consisted of around ten thousand malware files from 9 different malware families or classes. This was the training set. This project aims to predict the malware families of another set of ten thousand malware files, the test set.

Each file was provided in two formats. One was the *bytes* file (without header, in hexadecimal text) and the other was the *asm* file which was basically the information extracted by the IDA Pro disassembles on the bytes file. Each

```
text:00401000                   ; Segment type: Pure code
text:00401000                   ; Segment permissions: Read/Execute
text:00401000                   _text           segment para public 'CODE' use32
text:00401000                                   assume cs:_text
text:00401000                                   ;org 401000h
text:00401000                                   assume es:nothing, ss:nothing, ds:_data,
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@ABQBD@Z ; std
text:0040100D C7 06 08 BB 42 00                 mov     dword ptr [esi],offset off_42BB08
text:00401013 8B C6                             mov     eax, esi
text:00401015 5E                                pop     esi
text:00401016 C2 04 00                          retn    4
text:00401016                   ; ---------------------------------------------------------
text:00401019 CC CC CC CC CC CC CC              align 10h
text:00401020 C7 01 08 BB 42 00                 mov     dword ptr [ecx],offset off_42BB08
text:00401026 E9 26 1C 00 00                    jmp     sub_402C51
text:00401026                   ; ---------------------------------------------------------
```

Figure 1.1: Overview of an ASM file

malware file in the dataset had a unique *ID*, a 20-character *hashvalue* and a *Class*, an integer representing one of 9 family names to which the malware may belong. The 9 categories of malware are as follows:

1. Ramnit
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

```
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
00401090 8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010A0 08 50 51 52 56 E8 18 1E 00 00 83 C4 10 8B C6 5E
004010B0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010C0 8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010D0 08 50 51 52 56 E8 65 1E 00 00 83 C4 10 8B C6 5E
004010E0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010F0 33 C0 C2 10 00 CC CC CC CC CC CC CC CC CC CC CC
00401100 B8 08 00 00 00 C2 04 00 CC CC CC CC CC CC CC CC
00401110 B8 03 00 00 00 C3 CC CC CC CC CC CC CC CC CC CC
00401120 B8 08 00 00 00 C3 CC CC CC CC CC CC CC CC CC CC
00401130 8B 44 24 04 A3 AC 49 52 00 B8 FE FF FF FF C2 04
00401140 00 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
00401150 A1 AC 49 52 00 85 C0 74 16 8B 4C 24 08 8B 54 24
00401160 04 51 52 FF D0 C7 05 AC 49 52 00 00 00 00 00 B8
00401170 FB FF FF FF C2 08 00 CC CC CC CC CC CC CC CC CC
00401180 6A 04 68 00 10 00 00 68 68 BE 1C 00 6A 00 FF 15
00401190 9C 63 52 00 50 FF 15 C8 63 52 00 8B 4C 24 04 6A
004011A0 00 6A 40 68 68 BE 1C 00 50 89 01 FF 15 C4 63 52
```

Figure 1.2: Overview of a byte file

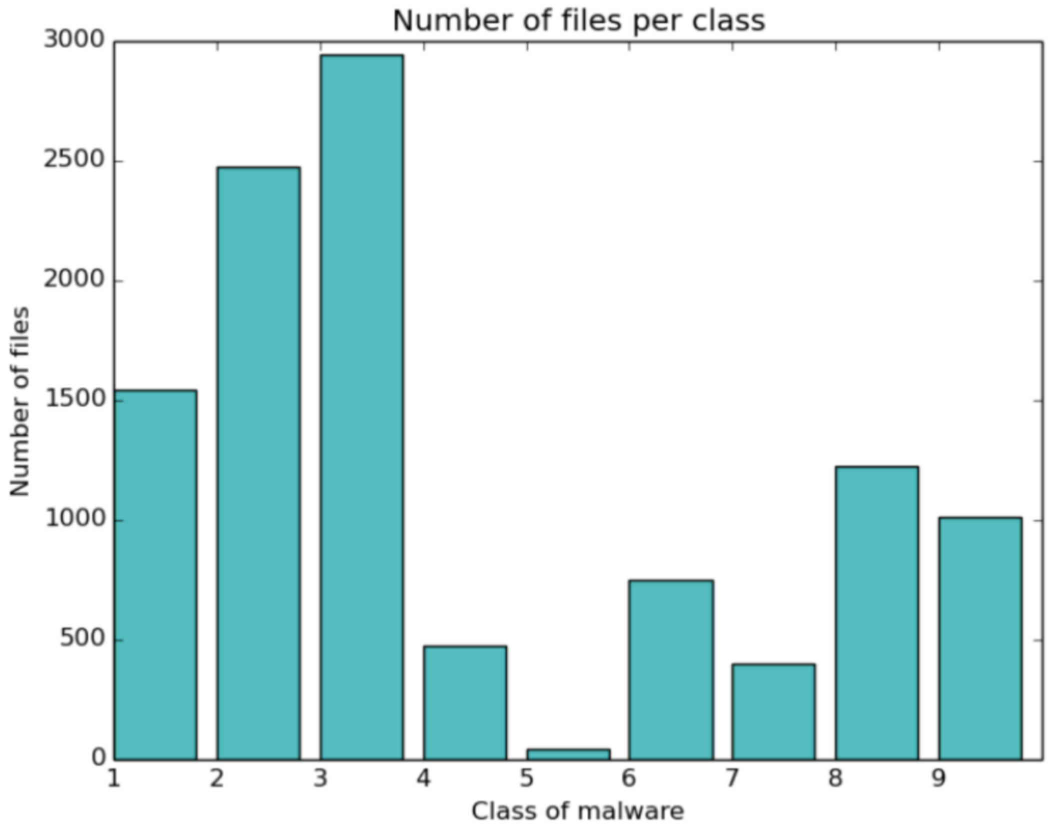

Figure 1.3: Class of malware vs Number of samples [3]

Apart from this, the dataset also contained the following files:

- train.7z - the raw data for the training set
- trainLabels.csv - the class labels associated with the training set
- test.7z - the raw data for the test set
- dataSample.csv - a sample of the dataset to preview before downloading

Malware identification dependent on machine learning strategies is regularly treated as an issue explicit to a specific malware family. In such cases, discovery includes preparing and testing models for each malware family. This methodology can for the most part accomplish high exactness, however it requires numerous grouping steps, bringing about a moderate, wasteful, and unfeasible process. Conversely, arranging tests as malware or benign dependent on a solitary model would be unquestionably progressively proficient. In any case, such a methodology is to a great degree complex and extracting common features from a variety of malware families might result in a model that is too generic to be useful.

Chapter 2

Models

2.1 Feature Extraction

After doing some research and study of the dataset, we felt like some of the simple file properties were quite useful to distinguish between the 9 malware classes. We used the file size of both bytes and asm files and even the compression rate of both the file types. Having this feature vectors gave us significant boost in accuracy and had a very high information gain.

Apart from this, we also created the TFIDF (term frequency inverse document frequency) for the contents in the ASM file over 2-grams. However, doing this for the huge dataset was not feasible, so we switched to the HashVectorizer, an efficient hash-based version of TFIDF which takes up less memory, time and space but also unfortunately reduces the accuracy slightly as well. The next feature we selected were the string file characteristics like urls, directories, registries, headers, entropy, etc. We combined all of these in a single vector that we then fed to our models.

2.2 Model Selection

2.2.1 Random Forest Classifier

This algorithm has been proposed by Breiman (2001) as an enhancement of Tree Bagging. According to Wikipedia, A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

At each test node the optimal split is derived by searching a random subset of size K of candidate attributes (selected without replacement from the candidate attributes)

2.2.2 ExtraTreeClassifier

An extra trees classifier (standing for extremely randomized trees) is a variant of a random forest. Unlike a random forest, at each step the entire sample is used and decision boundaries are picked at random, rather than the best one. In real world cases, performance is comparable to an ordinary random forest, but sometimes a bit better.

The Extra-Tree method was proposed with the main objective of further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree.

According to the paper, [5] on Extremely randomized trees,

> *The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other tree-based ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees.*

This thought is somewhat profitable with regards to numerous issues portrayed by a substantial number of numerical highlights changing pretty much consistently: it leads often to increased accuracy thanks to its smoothing and at the same time significantly reduces computational burdens linked to the determination of optimal cut-points in standard trees and in random forests.

2.2.3 LightGBM

There was another model we wanted to look at for our classification model, which was the LightGBM. LightGBM is a gradient boosting framework that uses tree based learning algorithm. It has been designed to be efficient with the advantages like faster training speed and higher efficiency, lower memory usage, better accuracy, support of parallel and GPU learning, capable of handling large-scale data, etc.

According to a Medium article [2], LightGBM grows tree vertically while other algorithm grows trees horizontally. Basically, in Light GBM the tree grows leaf-wise while in other algorithm the tree grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

2.3 Model Techniques and Training

There was no easy way to download the dataset locally. The uncompressed file size is close to half terabytes. For this purpose, we used an Cloud environment to provision a machine with adequate storage. Even after that, decompressing and performing any kind of operations on the entire dataset were taking hours. For this purpose, we divided the data equally to about ≈2000 files representative of the entire class population. Even after this reduction, the dataset size is close to ≈150GB in size.

We used ExtraTreesClassifier to classify the model because it gave us a quick overview of the results and had the highest performance on our reduced train/test dataset. Other classifiers we tested included Naive Bayes and Logistic Regression. In addition, Random Forests and Extra Trees train quickly compared to other gradient-based classifiers such as gradient boosted trees and multi-layer perceptrons. We also wanted to use a classifier which did multi-class classification which limited our options.

Initially, we deployed this model on the entire range of the HashingVectorizer from the .asm files. We attained an average accuracy of 72% on the small test set using 100 trees with a max depth of 5, and an average accuracy of 70%-75% on the small test set using 1000 trees with a max depth of 5.

After filtering down the features and using n gram modelling to ≈10000 top bigrams, we attained an average accuracy of 85%.

We then combined the asm features, bytes features and the metadata features in a single vector of ≈10108 features. This increased our average accuracy to ≈98%. While we have extracted the features significantly, we think this can be further reduced to increase the training time in the future.

We then tried using a newer model called LightGBM which has been gaining popularity recently. We used the same feature vector as before with hyperparmeters like $learningrate = 0.5$ and $boostingtype =' dart'$, and used the multiclass objective which gave us results matching our ExtraTreesClassifier model. Surprisingly, the LightGBM model was about 10x faster than ExtraTreesClassifier with the same accuracy and result of ≈ 98%.

Chapter 3

Discussion and Conclusion

Our final model, created using around 10,000 features on approximately 2000 malware files did end up giving us around 98% accuracy. Even though our features did a good job on the limited dataset we used, we feel, given the time, we could have tried other available features from the dataset to better train our model. For example, there's also a research work that describes how visualizing a malware file as an image could help in the classification task. We also feel we could improve our model by tuning and calibrating on the probabilities predicted by the model instead of directly using the produced probabilities.

We also realized via this project that being a subject matter expert on the data we're dealing with greatly helps in selecting and creating the best model as we could then wisely choose the important features from the set instead of trying a bunch of features and observing the output. Hence, the task of clever feature selection is interesting and challenging at the same time and offers a great deal of learning.

References

[1] Microsoft Malware Classification Challenge (BIG 2015) | Kaggle. (2015). Retrieved from https://www.kaggle.com/c/malware-classification/data

[2] What is LightGBM, How to implement it? How to fine tune the parameters?. Retrieved from https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc?fbclid=IwAR1BuUuh0yy-gZzhXqL2grT5ZTGHBhOqJfBzPTowO8ZBvApGWPdSdv5P9og

[3] rakeshchada/Microsoft-Malware-Classification-Kaggle. (2015). Retrieved from https://github.com/rakeshchada/Microsoft-Malware-Classification-Kaggle/blob/master/Project_Report.pdf

[4] Measuring the Effectiveness of Generic Malware Models https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?referer= https://www.google.com/httpsredir=1article=1566context=etd_projects

[5] (2006). Extra Tree Classifier http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.7485rep=rep1type=pdf