



INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Project Report on

MALWARE DETECTION

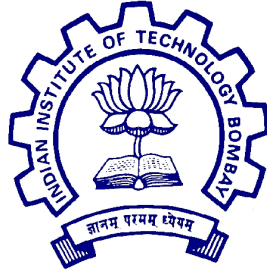
SUBMITTED TOWARDS THE
PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

**CS725:Foundations of Machine Learning (Computer
Science and Engineering)**

BY

Anurag Chaudhary	Roll No:193050061
Himanshu Aswal	Roll No:193059001
Pranav Chaudhary	Roll No:193059004
Sanyam Raj	Roll No:193050096

**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**



INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Project Entitled

MALWARE DETECTION

Submitted by

Anurag Chaudhary	Roll No:193050061
Himanshu Aswal	Roll No:193059001
Pranav Chaudhary	Roll No:193059004
Sanyam Raj	Roll No:193050096

is a bonafide work carried out by students and it is submitted towards the partial fulfillment of the requirement of CS725:Foundations of Machine Learning (Computer Science and Engineering).

Prof. Sunita Sarawagi
Department of Computer Science, IIT Bombay

Abstract

In recent years, the malware industry has become a well organized market involving large amounts of money. Well funded, multi-player syndicates invest heavily in technologies and capabilities built to evade traditional protection, requiring anti-malware vendors to develop counter mechanisms for finding and deactivating them. In the meantime, they inflict real financial and emotional pain to users of computer systems.

One of the major challenges that anti-malware faces today is the vast amounts of data and files which need to be evaluated for potential malicious intent. 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. One of the main reasons for these high volumes of different files is the fact that, in order to evade detection, malware authors introduce polymorphism to the malicious components. 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, such that they look like many different files.

In order to be effective in analyzing and classifying such large amounts of files, we need to be able to group them into groups and identify their respective families. In addition, such grouping criteria may be applied to new files encountered on computers in order to detect them as malicious and of a certain family.

Contents

1	Problem Statement	5
2	Goal and Objective	5
3	Data	5
4	Related Literature	7
5	Description of the set of approaches tried	8
6	Experiments	8
6.1	Code	8
6.2	Experimental Platform	8
6.3	Experimental Results	9
7	Effort	9
8	Conclusion	10

1 Problem Statement

In the past few years, the malware industry has grown very rapidly that, the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust softwares to detect and terminate these attacks. 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.

2 Goal and Objective

- Minimize multi-class error.
- Multi-class probability estimates.
- Malware detection should not take hours and block the user's computer. It should finish in a few seconds or a minute.

3 Data

Source : <https://www.kaggle.com/c/malware-classification/data>

For every malware, we have two files

.asm file

.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header).

.asm file		.bytes file
<pre>.text:00401000 gs:nothing .text:00401000 56 .text:00401001 80 44 24 00 .text:00401005 50 .text:00401006 8B F1 .text:00401008 E8 1C 1B 00 00 exception::exception(char const * const &) .text:00401000 C7 06 08 BB 42 00 .text:00401013 8B C6 .text:00401015 5E .text:00401016 C2 04 00 .text:00401016 ----- .text:00401019 CC CC CC CC CC CC CC .text:00401020 C7 01 08 BB 42 00 .text:00401026 E9 26 1C 00 00 .text:00401026 ----- .text:0040102B CC CC CC CC .text:00401030 56 .text:00401031 8B F1 .text:00401033 C7 06 08 BB 42 00 .text:00401039 E8 13 1C 00 00 .text:0040103E F6 44 24 00 01 .text:00401043 74 09 .text:00401045 56 .text:00401046 EB 6C 1E 00 00 .text:00401048 83 C4 04 .text:0040104E .text:0040104E 8B C6 .text:00401050 5E .text:00401051 C2 04 00 .text:00401051 -----</pre>	<pre>assume es:nothing, ss:nothing, ds:_data, fs:nothing, push esi lea eax, [esp+8] push eax mov esi, ecx call ??@exception@std@00AE@AB0BD0Z ; std:: mov dword ptr [esi], offset off_42B808 mov eax, esi pop esi retn 4 ; ----- align 10h mov dword ptr [ecx], offset off_42B808 jmp sub_402C51 ; ----- align 10h push esi mov esi, ecx mov dword ptr [esi], offset off_42B808 call sub_402C51 test byte ptr [esp+8], 1 jz short loc_40104E push esi call ???@YAXPAX@Z ; operator delete(vol add esp, 4 loc_40104E: mov eax, esi pop esi retn 4 ; -----</pre>	<pre>00401000 00 00 80 40 20 00 1C 02 42 00 C4 00 20 04 20 00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01 00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18 00401030 40 82 02 63 20 00 09 10 01 02 21 00 82 00 04 00401040 82 20 08 83 00 08 00 00 00 02 00 60 80 10 80 00401050 18 00 00 20 A9 00 00 00 00 04 78 01 02 70 90 00401060 00 02 00 08 20 12 00 00 40 10 00 80 00 40 19 00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00 00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00 00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00 004010A0 00 40 00 00 00 34 40 40 00 04 00 80 80 08 00 08 004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A 004010C0 06 01 02 00 40 00 00 00 00 00 20 00 02 00 04 004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82 004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00 004010F0 B4 00 00 40 00 02 29 25 08 00 00 00 00 00 00 00 00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00 00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00 00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10 00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11 00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10 00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01 00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00 00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00 00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11 00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00</pre>

Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files.

There are total 10,868 .bytes files and 10,868 asm files total 21,736 files

There are 9 types of malwares (9 classes) in our given data:

- Ramnit
- Lollipop
- Kelihos_ver3
- Vundo
- Simda
- Tracur
- Kelihos_ver1
- Obfuscator.ACY
- Gatak

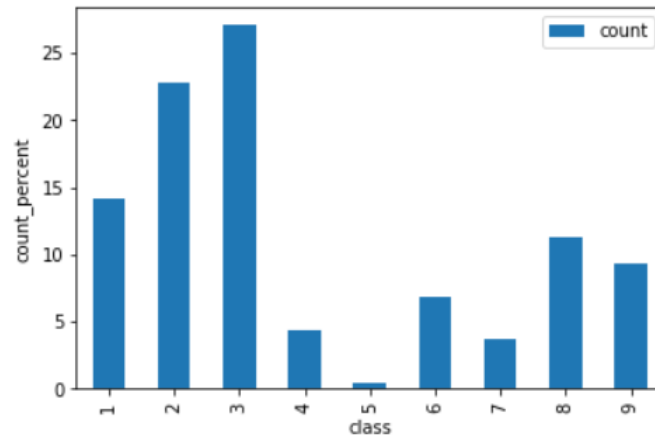


Figure 1: Data Distribution of various classes

4 Related Literature

- Microsoft Malware Winners' Interview: 1st place,"NO to overfitting!"
<http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/>
- Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification
<https://arxiv.org/pdf/1511.04317.pdf>
- First place approach in Microsoft Malware Classification Challenge (BIG 2015)
<https://www.youtube.com/watch?v=VLQTRILGz5Y>
- Malware Detection github
<https://github.com/dchad/malware-detection>

5 Description of the set of approaches tried

- Logistic Regression- We first tried with the Logistic Regression Model and using this model, accuracy of 76% was achieved.
- Random Forest- Using this model, 96% accuracy was achieved.

6 Experiments

6.1 Code

The code is developed in Python with the help of libraries mainly matplotlib, numpy, pandas and sklearn.

The code started by first of all visualising the data distribution among various classes.

The code can be found on this link: <https://git.cse.iitb.ac.in/pranavchaudhary/CS725>

6.2 Experimental Platform

The code was developed and tested on Windows using Jupyter Notebook. The code was run on a machine with configurations as:

- Processor: i7-9th Gen
- RAM: 16GB
- SSD: 256GB

The code ran on this machine for 60 hours(approx.)

6.3 Experimental Results

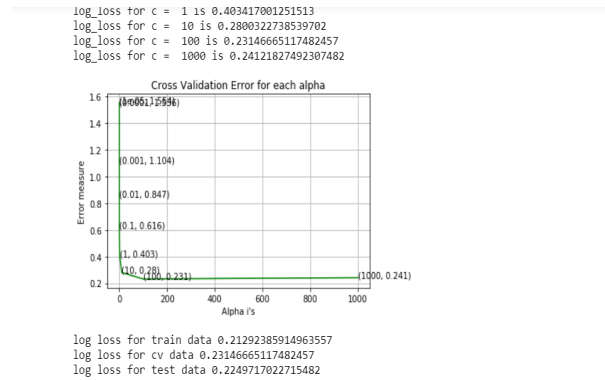


Figure 2: Logistic Regression Classifier Alpha vs. Loss Graph

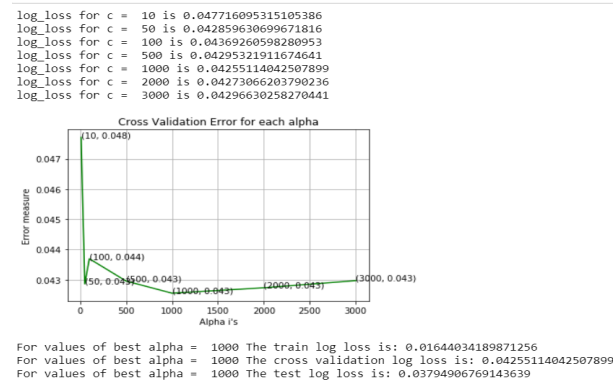


Figure 3: Random Forest Classifier Alpha vs. Loss Graph

7 Effort

The different parts of the project along with fraction of time taken by each part:

- Learning about Malwares and bytes and asm files-0.05
- Data Visualization-0.05
- Data Preprocessing-0.4
- Training-0.2
- Validation-0.2

- Testing-0.1

The most challenging and time taking part in this project was the pre-processing of dataset to a reasonable size without loss of information as the original dataset was large enough to train the model on our machines(around 184 GB). Fraction of work done by different team members:

- Anurag Chaudhary-0.25
- Himanshu Aswal-0.25
- Pranav Chaudhary-0.25
- Sanyam Raj-0.25

8 Conclusion

The dataset provided by Microsoft was of a very large size and had to be preprocessed using Feature Extraction to bring it to a size which could be run on our machines. The model was trained on bytes files as well as asm files using Logistic Regression Model and Random Forest Classifier Model. The results achieved by the Random Forest Model was much better as compared to the results achieved by the Logistic Regression Model.