MICROSOFT MALWARE CLASSIFICATION-

Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

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.
Source: https://www.avg.com/en/signal/what-is-malware

1.2. Problem Statement

In the past few years, the malware industry has grown very swiftly. Organizations now invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust software 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.**

1.3 Source/Useful Links

Microsoft has been very active in building anti-malware products over the years and it runs its anti-malware program on over **150 million computers** around the world. This generates tens of millions of daily data points to be analysed as potential malware. In order to be effective in analysing and classifying such large amounts of data, I need to be able to group them into groups and identify their respective families.

This dataset provided by Microsoft contains about 9 classes of malware. ,
Source: https://www.kaggle.com/c/malware-classification

1.4. Real-world/Business objectives and constraints.

1. Minimize multi-class error.

2. Multi-class probability estimates. I should be able to tell how much sure our model is of the estimate. If very sure, only then can I delete the malware.
3. Malware detection should not take hours and block the user's computer. It should finish in a few seconds or a minute. However, there is no strict low latency requirement.

---

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- Source : https://www.kaggle.com/c/malware-classification/data
- For every malware, I have two files

1. .asm file (read more: [https://www.reviversoft.com/file-extensions/asm)-](https://www.reviversoft.com/file-extensions/asm) asm represents the assembly language code. The compiler converts C language code to assembly code which is further converted to the bytes file (comprising of just 1s and 0s) by the assembler.
2. .bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

- Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:
- **Lots of Data for a single-box/computer.**
- 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 give data
- Types of Malware:

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

2.1.2. Example Data Point

.asm file (the commands are microprocessor commands)

```
.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@ABQBD@Z ; std::exception::exception(char
const * const &)
.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                    ; ---------------------------------------------------
-----------------------
.text:0040102B CC CC CC    CC CC                    align 10h
.text:00401030 56                      push    esi
.text:00401031 8B F1                      mov     esi, ecx
.text:00401033 C7 06 08    BB 42 00                 mov     dword ptr
[esi],    offset off_42BB08
.text:00401039 E8 13 1C    00 00                    call    sub_402C51
.text:0040103E F6 44 24    08 01                    test    byte ptr    [esp+8],
1
.text:00401043 74 09                      jz      short loc_40104E
.text:00401045 56                      push    esi
.text:00401046 E8 6C 1E    00 00                    call    ??3@YAXPAX@Z
; operator delete(void *)
.text:0040104B 83 C4 04                      add     esp, 4
.text:0040104E
.text:0040104E                    loc_40104E:              ; CODE XREF:
.text:00401043j
.text:0040104E 8B C6                      mov     eax, esi
.text:00401050 5E                      pop     esi
```

```
.text:00401051 C2 04 00                                retn    4
.text:00401051                          ; --------------------------------------------------
------------------------
```

.bytes file (encoded in hexadecimal)

```
00401000 00 00 80 40 40 28 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 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
00401060 00 02 00 08 20 12 00 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 08 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 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 20 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
```


2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

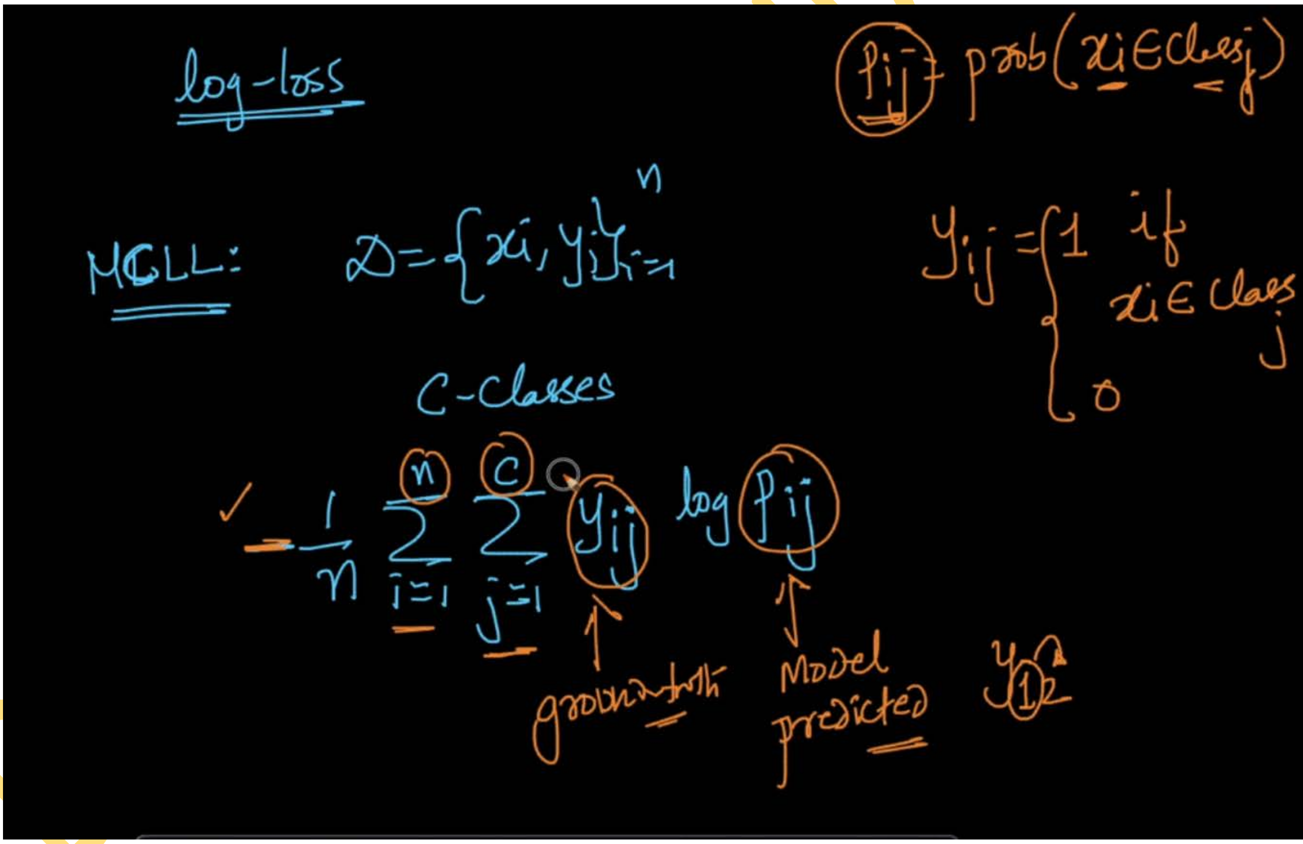
There are nine different classes of malware that I need to classify a given a data point => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/malware-classification#evaluation

Metric(s):

- Multi class log-loss



- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

* Class probabilities are needed.

* Penalize the errors in class probabilites => Metric is Log-loss.

* Some Latency constraints.

2.3. Train and Test Dataset

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

---

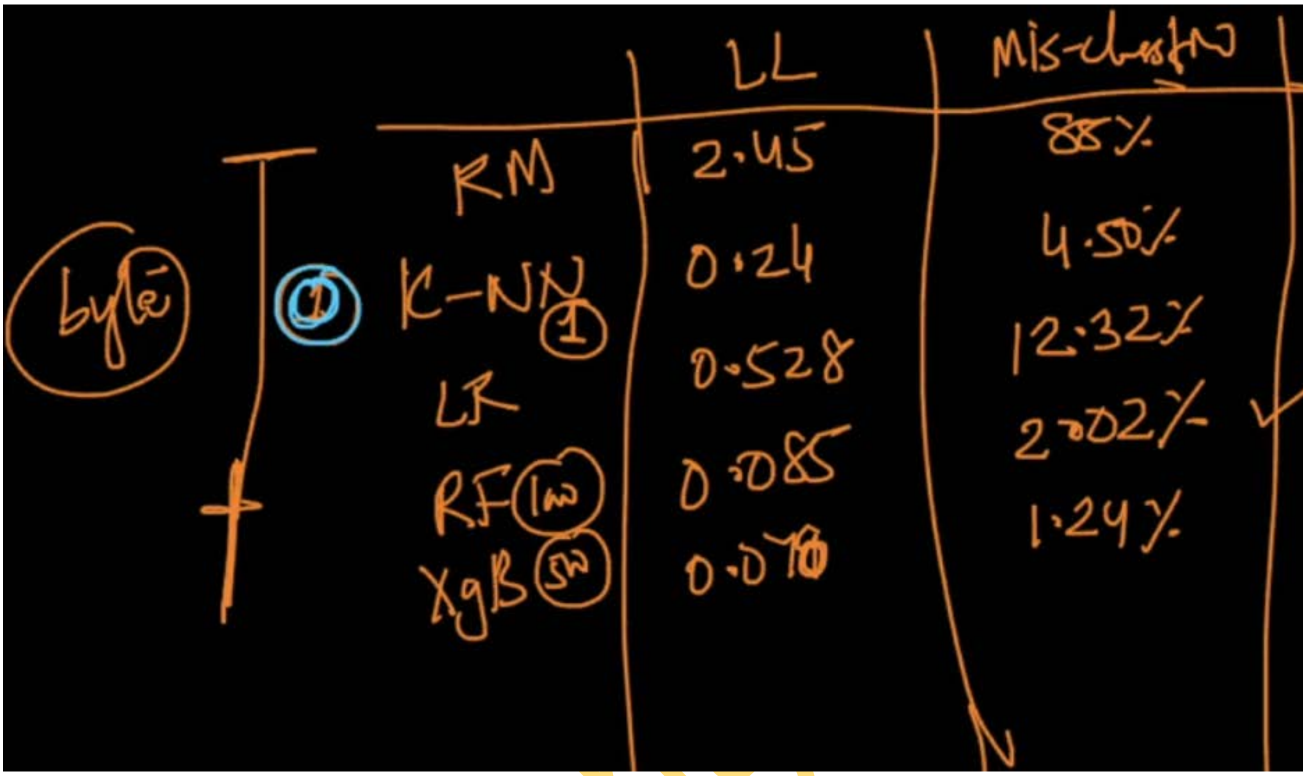
2.4. Useful blogs, videos and reference papers

---

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition:
https://www.youtube.com/watch?v=VLQTRlLGz5Y
https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

1. Countplot shows imbalanced dataset. Classes 4,5 7 occur less frequently and 1,2 more frequently.
2. EDA on bytes files is performed.
3. I perform a feature addition. Using os.stat, I add size of each bytes file as a feature. Box plot shows visible differences between IQR and max min of some classes.
4. The bytes files consist of an address followed by two hexadecimal digits. (=8 binary digits). Between 00-FF there are 256 distinct values (16*16).
5. The byte file contents are added to the corresponding id in the dataframe. The address is removed. And then CountVectorizer is applied.
6. Column normalization is performed.

7. Multivariate analysis using bytes file. T-SNE is performed on the BoW features. perplexity of 50 and 10 chosen. Typically, lower the perplexity, higher is the preference given to local features.
8. Clusters are formed. However, some clusters were overlapping but in general were differentiated well. Hence, the features are performing good. Clusters were stable with varying perplexity
9. 64-16-20 Train-CV-Test split is done. Distribution of classes is same in all
10. Our performance metric is log loss which has upper bound of infinity. So we generate a random model and calculate its log loss as a ceiling. 2.45 is the measured log loss.
11. The multiclass confusion and precision and recall matrices are plotted.
12. k-NN classifier is tried on the 257 bytes file features with 9 classes. Followed by calibrated classifier. Calibrated classification ensures that probability distribution estimate of classes by k-NN in some way reflects the underlying prob distribution in the dataset. It is required as kNN is a non-linear model.
13. Hyperparameter tuning is done using iteration through a python list and the best value of k is noted. it is 1. On calculating train log loss and cv log loss and test log loss, we find overfitting.
14. Confusion, precision, recall matrices plotted. Class 5 performance is relatively worse due to lack of data point.
15. Now, I use Logistic Regression with hyperparameters (10**-4 to 10**5). It does not classify Class 5 points at all. But no overfitting.
16. Random Forest Classifier with h-param as no. of trees (estimators) is used. No. of trees best=1000. The data is not VERY high-dimensional (approx. 1000) so can be used. Class 5 has 100 percent precision
17. XGBoost Classifier with varying n estimators is used. Best n_estimator=500. This classifier gave best results on train, cv, test log loss and matrices.
18. XGBoost with Randomized Search CV was employed.
19. Results of our models-



There are 10868 files of asm

All the files make up about 150 GB

The asm files contains :

1. Address

2. Segments

3. Opcodes

4. Registers

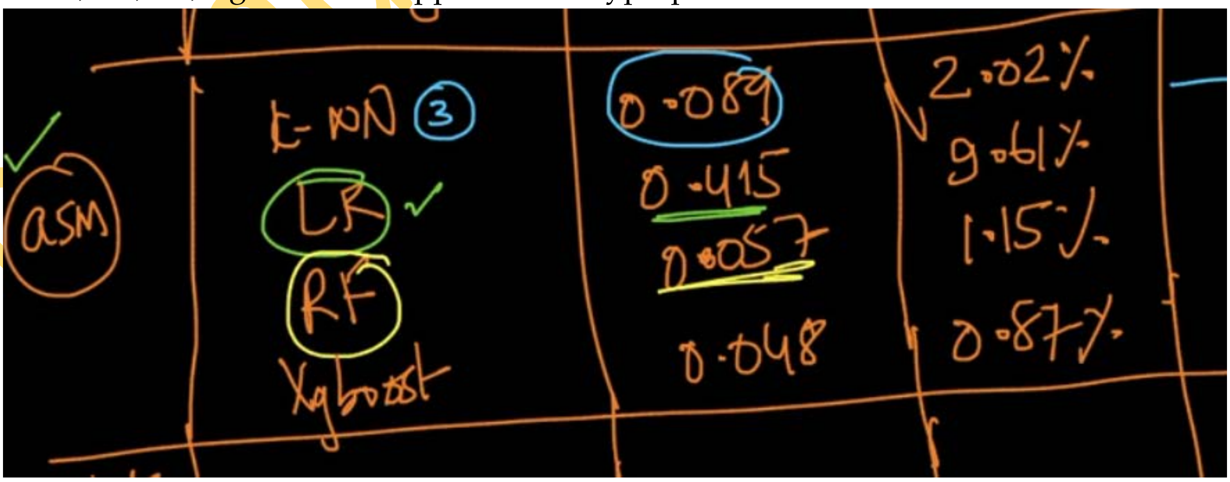
5. function calls

6. APIs

With the help of parallel processing we extracted all the features.In parallel we can use all the cores that are present in our computer.

Here we extracted 52 features from all the asm files which are important.

We read the top solutions and handpicked the features from those papers/videos/blogs.
Refer:https://www.kaggle.com/c/malware-classification/discussion

20.
21. The Asm features are extracted. There are 52 keywords in an Asm file. These are converted to BoW type model.
22. The Asm files are transferred to 5 folders to facilitate multithreading. Multithreading is when several functions are run in parallel into the RAM by several cores (4-8) of the CPU. multiprocessing.Manager() helps us attain this.
23. Now, we have BoW features of only the 52 keywords. A custom BoW was implemented with unigram features only as the keywords. A dictionary was mapped with a index to each keyword and an list was created with each index corresponding to the dictionary mapping. This was appended to the df alongwith a unique id.
24. Boxplot shows that size of asm files is a useful feature hence was added in the df.
25. A univariate analysis [boxplot] of some of the 53 features was performed to check usefulness. Features like .text were useful and some not.
26. Multivariate tSNE analysis did not cluster well.
27. KNN, LR, RF, XgBoost was applied with hyperparameter variation.
28. 

29. Now I combine asm+bytes features. (53+256 approx).
30. Multivariate tSNE shows good clusters.
31. Merging features gave the best RF and XGBCLassifier models.
32. Future improvements-

Add bi-grams and n-gram features on byte files and improve the log-loss

Using the 'dchad' github account (https://github.com/dchad/malware-detection), decrease the logloss to <=0.01
Watch the video (https://www.youtube.com/watch?v=VLQTRlLGz5Y)and implement the image features to improve the logloss </