Microsoft Malware Classification Challenge

Rakesh Chada
Stony Brook University
rchada@cs.stonybrook.edu

Nitish Gupta
Stony Brook University
nitigupta@cs.stonybrook.edu

ABSTRACT
One of the major challenges that the anti-malware softwares face these days is handling the vast amount of files that needs to be scanned. This vast amount of data is majorly due to the polymorphism introduced by the malware authors. To effectively analyze these files, it would be helpful to identify all malware files that belong to the same family(class). In this report, we present the techniques and results of our attempt to perform such a classification of malware files into their respective classes.

Keywords
Malware, Multi-class classification, Logistic Regression, Random Forest, Feature Selection, Log Loss

1. INTRODUCTION
The volume of malware is huge and growing quickly. As of 2013, Microsoft receives over 150 thousand new, unknown files each day to be analyzed[2]. One of the reasons for these high volumes of different files is due to polymorphism introduced by malware authors. Due to these obfuscation tactics, malware files belonging to the same class may look like different files. In order to be effective in analyzing these malware files, we must be able to classify these files into the class they belong to. The classification may be applied to new files encountered to detect them as malicious and of a certain family. Given the volume of the task, anti-virus companies utilize machine learning techniques to detect and classify malware.

The goal of our project is to classify the given malware files into 9 different classes. Instead of doing a hard labeling, we predict the probabilities of belonging to each class (soft labeling). This task is posed as a challenge in kaggle by Microsoft[1].

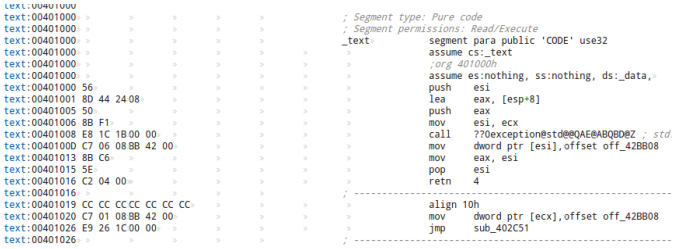
2. PRIOR WORK
There is a fair amount of existing work on Malware classification. There are different ways of classifying malware.

Signature Based Detection: Malware has a certain specific pattern of bits which is unique and forms its signature. Anti - malware softwares based on this method scan for these patterns to classify malware. However, there must be an up to date database of signatures. With simple code obfuscation techniques, malware may avoid detection through this method[3].

A few approaches [4], [5] to classify malware involved examining the structure of the assembly files. These structural features of executables are likely to indicate the presence of inserted malicious code. Another approach is opcodes statistical-based classifier based on decision tree [6]. Hidden Markov models [7] have been applied to this problem and have also been proved to show good results in classifying malware.

3. DATA DESCRIPTION
The data is provided by Microsoft on Kaggle website. The size of the data is 500GB(uncompressed). We have 10868 test samples. Each sample has its corresponding .asm and .bytes file. The .bytes file is a hexadecimal representation of the binary content of the file. These files are generated using IDA Disassembler tool. We have 10373 training samples. Each sample belongs to one of the 9 classes of malware namely Ramnit, Lollipop, Kelihosver3, Vundo, Simda, Tracur, Kelihosver1, ObfuscatorACY, Gatak. The snapshots of the .asm and .bytes files are provided below.

Figure 1: Snapshot of an .asm file



4. EVALUATION
The evaluation metric for all our Machine Learning models is log loss. This is calculated as below:

$$Logloss = \frac{1}{N} * \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} * \log p_{ij}$$

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.

A model with a lower value of logloss is preferred to a model with a higher value of logloss.

Figure 2: Snapshot of a .bytes file

```
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
```

We also present our standing in the Kaggle's leaderboard for each of the model we had built.

5. MODELS

5.1 Baseline Model

Our initial Baseline Model was to predict the same probability for each class. So, we predicted $1/9 = 0.111$ for each class and for each malware file. This resulted in a log loss of 2.197. We stood 322nd (out of 377) in the Kaggle leaderboard with this model.

We improved our Baseline Model by predicting the actual class's probability instead of using the same probability for each class. We calculate this just by counting the number of instances of that particular class and then dividing by the total number of instances. This gave us a log loss of 1.89. We stood 302nd (out of 377) in the Kaggle leaderboard with this model.

5.2 Exploratory analysis

Given the huge amount of data, we did some exploratory analysis to gain a better understanding of the data. We did the following analysis:

- How does the class frequency distribution look like?
- How does each class differ in terms of file sizes?
- Prior research[5][6] suggests that ratio of opcodes is similar in malwares belonging to the same class and differs across classes. So, we were interested in finding if this holds for our dataset.

Fig. 3 shows the class distribution of each of the malware families. We can infer that there is a non-uniform distribution of the classes. Class 3 is the most frequent while Class 5 is the least frequent class. Fig. 4 shows a scatter plot of .asm and .bytes file sizes with colors of the circles representing various malware families. We can infer that certain

Figure 3: Class of malware vs number of samples

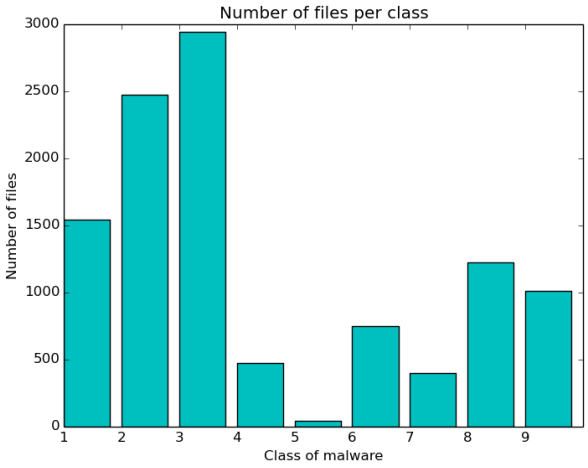


Figure 4: File sizes vs class

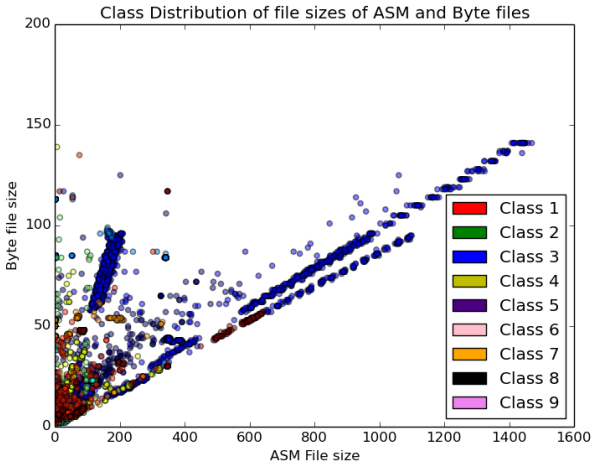
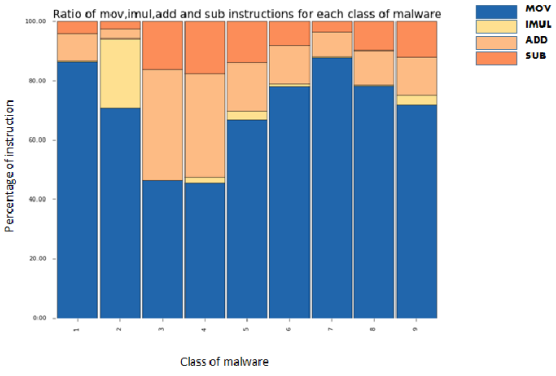


Figure 5: Ratio of opcodes for each class



classes are distinguishable by their file sizes. For instance, class 1 tends to have lower .bytes and .asm file sizes. Fig. 5 shows the plot of the frequency ratios of the top 4 frequent asm instructions per class. We can infer that this ratio is unique for few of the classes (for instance class 1 vs class 3) and hence this would serve as an useful feature in classification. Though we had plotted for only the top 4 frequent features, we believed that it would be interesting to use the frequencies of all 147 opcodes and see if they would help increase the classification accuracy.

5.3 Advanced Models

Once we had the Baseline Model, we built several other Machine Learning models that were trained using features from the dataset. Most of the features for our models have a common underlying theme of using the "frequency". We had used features such as the frequency of opcodes of several asm instructions, frequency of system calls, file sizes of .asm and .bytes files. As our final model, we had used an ensemble of these features to evaluate if it results in a more accurate prediction.

5.3.1 File Size Model

From our exploratory analysis on the data, we had discovered that the file sizes of .bytes and .asm files would be useful features to incorporate in the prediction of class family. So, as our first model, we had just used these binary features (.bytes and .asm file sizes) and trained a Logistic Regression Model. We achieved a log loss of 1.18 with this model. We stood 285th (out of 377) on Kaggle Leaderboard with this model. This is a improvement over the Baseline but we thought we could do better by incorporating additional features.

5.3.2 Opcode Frequency Classifier

We had discovered from our initial research paper readings and from our exploratory analysis that opcode frequency of certain ASM instructions play a key role in identifying malware family. So, we had extracted counts of each ASM instruction in all training and test files. We had around 147 total different ASM instructions. Using these 147 frequency features, we first trained a Logistic Regression Model. We had used a 10-fold cross-validation to determine the value of the regularization parameter 'C' that results in the optimal log loss. We achieved a log loss of 1.09 with this model. This is an improvement over both the Baseline and the File Size Models.

However, Logistic Regression is a linear classifier and wouldn't do a good job when the dataset is not linearly separable. With the 147-dimensional dataset, we were not sure if our dataset would be linearly separable. This led us to the idea of using an ensemble model that combines multiple linear separators to eventually result in a non-linear classification. So, we had trained a Random Forest model which uses an ensemble of Decision Trees. Again, we had used 10-fold cross-validation to determine the value for parameter "numEstimators" that determines the number of decision trees to fit. We achieved a log loss of 0.058 with this model. This is by far the best log loss we achieved till this point. We stood 168th (out of 377) on Kaggle Leaderboard with this model.

5.3.3 Random Forest with Feature Selection

We had observed that it takes an enormous amount of time to train a Random Forest Model on 10373 train data files with 147 features. So, we planned to reduce the size of the feature set by using only the most important features. The library "sklearn.ensemble.RandomForest" in Python has a built-in feature ranking mechanism which outputs a list of features with their ranks. This works based on the "Information Gain" criteria. The top ranked features would be the ones which would result in the best Information Gain when split using them. So, using this measure, we extracted the top 10 scoring features among the opcode frequencies. We now trained a Random Forest model using just these 10 features. We had achieved a log loss of 0.07 with this model. Though this is slightly higher than the previous log loss of 0.058, we believe that it's a good trade-off for the computational time saved. We stood 190th (out of 377) on Kaggle Leaderboard with this model.

5.3.4 System Call Frequency Model

Each malware class has distinct functions. For instance, Ramnit (Class 1) tries to disable the firewall of a computer, Vundo (Class 5) is an adware that generates popups and advertisement, Kelihos is a bot which sends spam messages.These tasks may require differing number and types of syscalls to achieve.

Therefore, we thought the number of syscalls could be a distinguishing feature. We integrated syscall counts with Opcode Frequency Model and trained a Random Forest classifier. However, this actually increased the log loss of our model to 0.27. So, this new feature might have added noise to the data and hence including this seemed redundant. However, a model which uses the actual syscalls (instead of the frequency) as features may be a direction worth exploring.

5.3.5 Ensemble Model

Once we had tested our model using different sets of features individually, our final idea was to use an ensemble of all these features and train a Random Forest Model to evaluate if it results in a better accuracy. So, we had combined the file size features and the opcode frequency features (top 10) and trained a Random Forest Model. We achieved a log loss of 0.049 with this model. This is the best log loss we had achieved among all the models. We stood 151st (out of 377) on Kaggle leaderboard with this model.

5.4 Per-class Accuracy

Although we had been using log-loss as our standard measure of evaluation, we thought it would be interesting to evaluate the accuracy of our model when we predict "hard labels" for the classification task. The hard labels are predicted by choosing the class that has the maximum probability value. The per-class accuracy of cross-validation on the training set for the Random Forest model is as below:

Class1	Class2	Class3	Class4	Class5	Class6
96.39%	99.71%	99.89%	97.48%	91.89%	96.98%

Class7	Class8	Class9
99.5%	98.57%	99.6%

We can infer that the classifier is most accurate on Class 3 and least accurate on Class 5.

5.5 Results Summary

The table below summarizes the results of all the models we had used for our classification task. We also present the Kaggle Leaderboard standing for each of our models and also the cross-validation accuracy on the training data.

Model	Log Loss	CV Accuracy	Kaggle Leaderboard
Baseline	1.89	26.62%	302/377
File Size	1.18	83.16%	285/377
Opcode Frequency (all features)	0.058	98.14%	168/377
Opcode Frequency (top 10 features)	0.07	96.28%	190/377
Ensemble (File Size + Opcode Frequency)	0.049	98.93%	151/377

6. CONCLUSION AND LEARNINGS

- The major difference that we had observed between our models and the models of the top rankers is that they had done a lot of calibration and tuning on the probabilities predicted by the model instead of directly using the produced probabilities.
- Apart from the above difference, the types of features and the types of models didn't differ too much in terms of the performance between our respective submissions.
- Feature selection is crucial to the performance of the Machine Learning model. An exploratory analysis of the data provides valuable insights in this aspect.
- Using redundant and noisy features might reduce the classification accuracy. For instance, when we tried to integrate "System Call Frequency" with our existing models, our log loss had increased.
- An interesting thing that has turned up as part of our project is the trade-off between computational time and the accuracy of the model. When we had used around 150 features and trained a Random Forest Model with around 1000 decision trees, it took around half of the day to train the model on our entire dataset. This gave us a log loss of 0.052. However, when we had used just the top 10 features and trained a Random Forest Model with the same 1000 decision trees, it just took a few hours for the training to complete. The log loss had slightly increased to 0.07. The amount of difference that this makes should probably be attributed to an individual's perspective. It might be a significant difference for a few and it might be a good trade-off for the rest. For us, it seemed like a good trade-off.

7. FUTURE WORK

Even though our features did a fairly good job in the classification task, we could still try to incorporate more interesting features especially considering the fact that most of our features relied on frequency. For instance, there's a research work[8] that shows how visualizing malware files as images would help in the classification task. So, we could construct such "image-pixel" like features from the .asm and .bytes files. They might prove to be useful in distinguishing the malware families. Furthermore, for the system call frequency feature that we had used, it might be worth exploring to evaluate if it leads to a better log loss if we use the actual system calls as features instead of using the frequencies. We can do this by constructing a bit vector of all distinct system calls (there are just around 50 system calls) and then setting the corresponding bits for the system calls being invoked in the file. Then, we can use the individual bit as a feature. The header information has also been used by several top rankers who claim that it is an excellent feature. For instance, the 11th line of the header had been identified as a good distinguishing feature[9]. This feature is of special interest as it looks only at a small subset of the data to do the classification. So, this task of clever feature selection is an interesting and challenging one and there's a great deal to learn in this area.

8. REFERENCES

[1] www.kaggle.com/c/malware-classification

[2] Dahl, George E., et al. "Large-scale malware classification using random projections and neural networks." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[3] Annachhatre, Chinmayee, Thomas H. Austin, and Mark Stamp. "Hidden Markov models for malware classification." Journal of Computer Virology and Hacking Techniques (2014): 1-15.

[4] Weber, Michael, et al. "A toolkit for detecting and analyzing malicious software."Computer Security Applications Conference, 2002. Proceedings. 18th Annual. IEEE, 2002.

[5] https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Bilar.pdf

[6] Rad, Babak Bashari, et al. "Morphed virus family classification based on opcodes statistical feature using decision tree." Informatics Engineering and Information Science. Springer Berlin Heidelberg, 2011. 123-131.

[7] Austin, T. H., Filiol, E., Josse, and Stamp, S. M. âĂIJExploring Hidden Markov Models for Virus Analysis: A Semantic Approach, Proceedings of the 46th Hawaii International Conference on System Sciences, Wailea, HI, USA, 2013, Jan 7-10, 50395048

[8] Nataraj, Lakshmanan, et al. "Malware images: visualization and automatic classification." Proceedings of the 8th international symposium on visualization for cyber security. ACM, 2011.

[9] https://www.kaggle.com/c/malware-classification/forums/t/13509/brief-description-of-7th-place-solution/72485