Microsoft Malware Classification Challenge

Classifying binary and assembly malware files into families

Vedasagar Karthykeyan
Johns Hopkins University
vkarthy1@jhu.edu

Yong Wu
Johns Hopkins University
ywu118@jhu.edu

Arvind Arikatla
Johns Hopkins University
Arvind.Arikatla@jhu.edu

ABSTRACT

This project attempts to solve a common security challenge proposed as a challenge by Microsoft. The challenge attempts to mimic real-word scenarios of cybersecurity through having participants work on the ever-growing challenge of classifying given malware into separate families in order to gain knowledge on how to deal with the malware. We approached this challenge by reading popular research papers on malware security as well as studying the performance of other groups who have joined in on the challenge and have shared their answers publicly. We created a model formed from a feature set we saw best fit from what we learned and we were able to get a high accuracy in detecting malware, although still below the top competition in the challenge.

Keywords

Malware Classification, Machine Learning, Feature extraction

1. INTRODUCTION

Early methods used signature based detection to detect malwares such as those used by Anti-Virus software. But these methods cannot be used anymore as malware authors introduce new techniques such as polymorphism to obfuscate malicious executables in order to avoid detection. Malware belonging to the same family are modified and hidden and thus signature detection schemes cannot be applied anymore [5]. Thus, researchers have started using heuristics and content based approach to detect them in recent years. Content-based approach analyzes the executables and tries to extract the most useful features that can be used to classify them as malwares. Using these features, researchers have been able to group malwares into their respective families.

In this project, we attempt to build our own feature extraction model to classify malwares into different families. We first extract the most discriminative features using domain knowledge and select the best features from the entire feature set. Finally, we train the different types of features sets using several classifiers and select the one that gives us the best accuracy for the selected set of features.

2. RELATED WORK

There has been significant work in recent years to detect malware based on their raw assembly files and content. Shabtai et al. [1] talk about using data-mining techniques to create accurate detection mechanisms for binary files. They extract byte n-grams that is byte sequences of n characters extracted from the binary code. Rad et al. [2] implement a statistical feature extraction model by using the frequencies of opcode and calculating the dissimilarity between different files. Schultz et al. [4] explore a number of data mining techniques to compute accurate detection mechanisms for unseen executables and binaries. They use features like DLLs used in the library, different function calls in the files and also the frequency of different GNU strings appearing the raw files. Masud et al. [3] use a scalable, multi-level hybrid feature extraction model to classify the malwares. They use features like binary n-grams, derived assembly features and dynamic link library calls and combine them into a hybrid feature retrieval model. They use the assembly features like the flow and series of function calls to determine the family of the malware file.

3. DATASET

The dataset for the classification challenge has been provided by Microsoft. The dataset consists of assembly and binary code for around 20,000 malicious files. Each malware file is identified by a 20-character hash and is also assigned the class to which it belongs in a separate csv file. There are 9 malware families to which each of the file can belong to. These 9 families are:

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

The entire dataset (including both training and testing files) was about half a Terabyte. Since we did not have the sufficient resources to extract and train features on a data this huge, we just picked up 500 files of each class in the training dataset.

4. OUR IMPLEMENTATION

The main task of our project was to identify the most discriminative malware features and extract these features from the given assembly and byte code. After successfully extracting the identified features, we trained the features on several classifiers and selected the one which gives the best accuracy. We also applied cross-validation on the final classifier in order to avoid overfitting and to get a more realistic result for our experiment.

Due to the large dataset and limited computational resources we started small and then trained on larger files. We first just extracted 50 files for each malware family. We then extracted all the features from these 50 files. This finally resulted in around 7000 features. We selected the best features out of entire feature set, selecting the ones that gave us the best information gain.

This entire process was repeated for 200 files of each family and again we compared the results with the previous results. After this we came up with the final feature set based on the above 2 iterations.

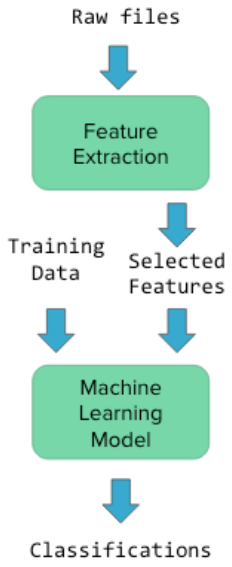


Figure 1: Our approach to build the classifier

We finally extracted 500 files (around 55 GB) and extract the features from them based on the final feature set we determined. The initial features extracted includes:

- Opcode frequency
  - More than 600 opcodes, such as mov, push, etc.
- Opcode index frequency
  - 257 opcode indexes (00, 01, 02 …, FD, FE, FF, ??)
- Dynamic Link Library (DLL) call features
  - Corresponds to a DLL function call in an executable
  - E.G. 'SHLWAPI.dll', 'OPENGL32.dll', 'CRYPT32.dll'
  -

The final feature set we got based on information gain:

- Numerical Features
  - 'file_size_ratio' – ratio of assembly file size to the byte file size
  - 'loc_count', 'header_count', 'text_count', 'rsrc_count', 'rdata_count', 'data_count', 'code_count', 'bss_count' – number of lines for different sections of the assembly files
  - 'mov', 'push', 'call', 'add', 'pop', 'cmp', 'lea', 'and', 'test', 'jmp', 'imul', 'mul', 'jz', 'jnz', 'or', 'adc', 'sub', 'xor', 'inc', 'dec' – 20 most frequent assembly instruction calls
  - '??', '00', 'FF' – Frequency of bytes in the byte code
- Binary features
  - .dll calls – dynamic link library calls made
  - std calls – standard library calls made
- Combined feature set
  - Numerical features + binary features

We then trained the 3 types of feature sets on different classifiers and selected the one which gave us the best results for cross-validation. We split the data into training data (75%) and testing data (25%). The classifies used by us are:

- K-Neighbors
- Decision Tree
- Random Forest
- SVM
- Logistic Regression
- Naive Bayes
- Quadratic Discriminant Analysis

5. EXPERIMENT RESULTS

5.1 For Numerical Data

- accuracy of "K-Neighbors" model is 0.923391215526
- accuracy of "Decision Tree" model is 0.97037793667
- **accuracy of "Random Forest" model is 0.992849846782**
- accuracy of "SVM" model is 0.208375893769
- accuracy of "Log Reg" model is 0.905005107252
- accuracy of "Naive Bayes" model is 0.524004085802
- accuracy of "QDA" model is 0.91215526047

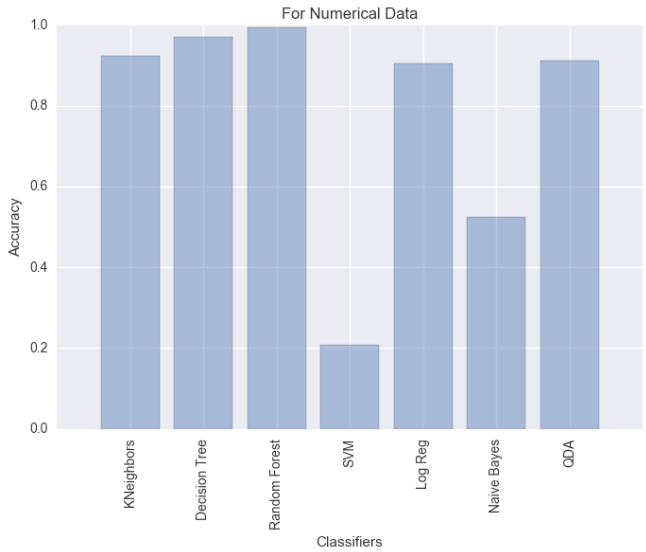


Figure 2: Comparison of performance of different classifiers on the numerical feature set

5.2 For Binary Data

- accuracy of "K-Neighbors" model is 0.819203268641
- accuracy of "Decision Tree" model is 0.872318692543
- accuracy of "Random Forest" model is 0.880490296221
- accuracy of "SVM" model is 0.880490296221
- **accuracy of "Log Reg" model is 0.884576098059**
- accuracy of "Naive Bayes" model is 0.800817160368
- accuracy of "QDA" model is 0.55873340143

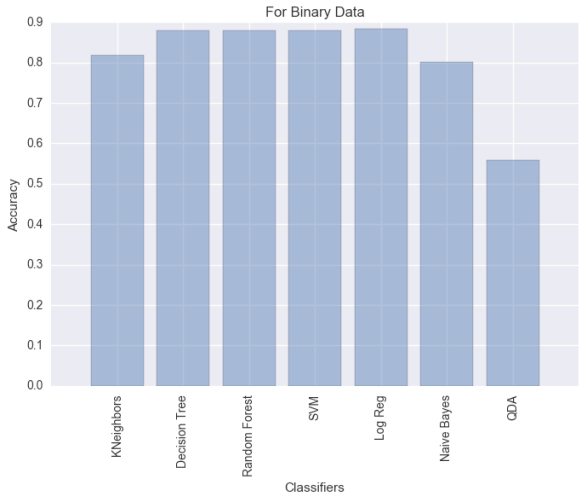


Figure 3: Comparison of performance of different classifiers on the binary feature set

5.3 For Combined Data

- accuracy of "K-Neighbors" model is 0.923391215526
- accuracy of "Decision Tree" model is 0.97037793667
- **accuracy of "Random Forest" model is 0.990806945863**
- accuracy of "SVM" model is 0.208375893769
- accuracy of "Log Reg" model is 0.902962206333
- accuracy of "Naive Bayes" model is 0.524004085802
- accuracy of "QDA" model is 0.91215526047

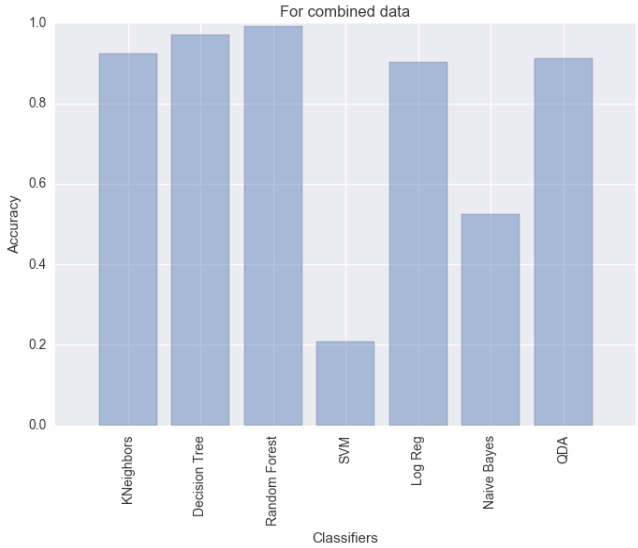
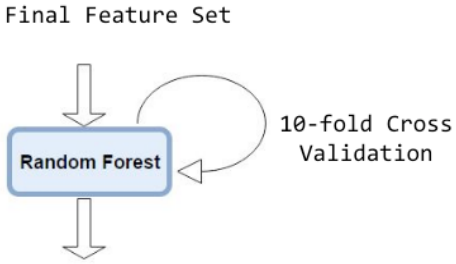


Figure 4: Comparison of performance of different classifiers on the combined feature set

Based on the above results, we can clearly see that Random Forests performed the best overall. Thus, we selected it for cross-validation. We used 1-10 folds of cross-validation and calculated the mean accuracy score. The results were as follows:

- accuracy is 0.864450127877 with fold 1
- accuracy is 0.979539641944 with fold 2
- accuracy is 0.971867007673 with fold 3
- accuracy is 0.982097186701 with fold 4
- accuracy is 0.994884910486 with fold 5
- accuracy is 0.864450127877 with fold 6
- accuracy is 0.971867007673 with fold 7
- accuracy is 0.953964194373 with fold 8
- accuracy is 0.918158567775 with fold 9
- accuracy is 0.976982097187 with fold 10
- **mean accuracy is 0.947826086957**



0.947826086957

6. LIMITATIONS

Although we got a high accuracy score, we weren't on par with the accuracy of the top 10 teams who have participated in the Microsoft Malware Challenge. There were several reasons we were limited to the accuracy we had. The first being time; we had a marginal amount of time and people devoted to the project whereas the actual challenge lasted much longer for the teams to perform research on. Although we did have the advantage in studying other group's projects and analyzing the precedence set by them. But perhaps the biggest limitation set forth on us was our limited resources in computational machinery.

All of the top teams had access to a computer with over 100GB of RAM and multiple high performance processors. We were limited to our laptops and computers from the lab. The main reason this was important was because several data analytic methods such as n-grams required very high computation and was a main component in many of the top team's models. Had we had access to such machinery, we would have no doubt been able to implement such methods to improve our accuracy.

7. CONCLUSIONS

Signature based detection mechanisms cannot be used today to detect malicious executables in the wild. There is an imminent need for coming up with more robust techniques to identify

malwares. This can be achieved using heuristics and analyzing the content and characteristics of the files.

Coming up with a feature set forms the main task of classifying data. Researchers need to come up with the most useful features using domain knowledge and extract these features from the raw files. The process has to be repeated in multiple iterations to get the most realistic accuracy. This can be achieved using methods like cross-validation to avoid over-fitting. Random Forests performed the best overall giving really high accuracies for all kinds of data. We finally achieve a mean accuracy of 94.78% to successfully classify the given malware files into 9 separate families.

8. REFERENCES

[1] Robert Moskovitch "Detecting unknown malicious code by applying classification techniques on OpCode patterns" Springer-Verlag "http://link.springer.com/article/10.1186%2F2190-8532-1-1", 2012, pp. 1-22.

[2] Babak Bashari Rad, Maslin Masrom. Metamorphic Virus Detection in Portable Executables Using Opcodes Statistical Feature. Proceeding of the International Conference on Advanced Science, 2011

[3] L. K. Mohammad M. Masud and B. Thuraisingham. A scalable multi-level feature extraction technique to detect malicious executables. Information Systems Frontiers, 2007.

[4] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001.

[5]"https://www.kaggle.com/c/malware-classification", [ONLINE], 2015