MALICIA - Malware Classification

Manika Mittal
Department of Computer Science
University of California, Los Angeles
manika.mittal@cs.ucla.edu

Ronak Sumbaly
Department of Computer Science
University of California, Los Angeles
rsumbaly@cs.ucla.edu

Abstract

The continuous rise in Malware attacks have led to the development of many Malware Detection Systems which use several techniques to identify known and unknown malwares. However most of these systems identify based on some previously known malware "signature". In order to escape detection the authors of malware have started obfuscating the code. This project presents a technique which uses Machine Learning to classify different malwares into their respective families. The main reason behind using Machine Learning is that while a code can be obfuscated using techniques like Garbage Code Insertion and Instruction Permutation, since at the lower level a malware, belonging to one family, performs similar functions it has to generate similar opcodes and similar patterns. We present a detailed analysis of various classification algorithms that can successfully be used to classify Malware by identifying these similar patterns.

1 Motivation

Malicious Software a.k.a Malware is a code snippet or an intrusive software which is written with an intent to disrupt computer operations, gather sensitive information from systems, inject computer viruses, trojans etc. While several schemes have been proposed to detect malware, obfuscation and polymorphism is used in order to elude the detection mechanisms, thus rendering the standard practices of identifying similarities/patterns in the malware software useless. Introducing these techniques means that the malware belonging to the same families, with similar forms of behavior are being changed constantly to look different. Thus we need to rely on analysis of the binaries that are generated by malwares, to classify the malwares into different families and hence detect them. And with an increase in the malware attacks, this classification has crucial applications in the world of security. This is the motivation on which our project has been proposed.

2 Background

As technology increasingly becomes an inseparable part of our lives, it has opened several new windows of opportunities for both people and criminals. Crime has shifted from the physical world to the virtual world, where the criminals exploit technology to launch attacks and harm people's privacy and data. This shift has amplified the effect of an attack, because now the attackers can attack from anywhere at any time and target much more people than it could have in the physical world. Several malware attacks have been witnessed lately, including attacks on Yahoo advertising server, Apple OS X, etc. This calls for an effective way to detect the malwares before they are launched so that appropriate steps can be taken to protect a system against such attacks.

Excellent technology already exists in the field of detection of malicious executable in the form of various anti-virus softwares such as Avast, Norton etc. All these anti-viruses use signature based detection where they check the contents of a file against a dictionary of known virus signatures.

Although this approach can effectively contain the known virus outbreaks, this technique fails when the system encounters an unknown virus and thus analysis of binaries, generated by malware, forms a more reliable means of malware detection. Since analysis of binaries is a general field it is host to wide range of approaches. In literature most of the work follows the framework of extracting features from malware binaries, applying a machine learning system to generate a model to classify malware and finally analyzing aspects of scaling up the framework to identify malware for large data-sets.

Most of the data-sets previously used to construct the classification model only comprised of the malware binaries. In this project we are also considering the ASM file generated via the IDA disassembler tool, giving us more ground to find patterns. Feature extraction in the existing models is based on manual selection of unique features which were identified by "looking" at the binary files. Most of the models have not employed a systematic machine learning approach to extract the frequently occurring features. The few who have, have intuitively applied the n-gram model [1], with n = 2, 4, 10 etc. The selection of n did not have any machine learning basis. We use frequent item-set approach which helped us construct a solid framework to use various classification algorithms.

Previously employed machine learning systems, are Gradient Boosting [2], Random Forest, SVM [3], Naive Bayes [4] and Decision Trees [5] which have achieved comparable accuracies. We ran several classification algorithms on the features that we generated in order to compare the various accuracies. Not only did we compare the accuracies of several classification models, we also compared how our feature selection was incrementally improving the accuracies of the same classification algorithms on the same dataset. Our main focus was feature extraction because we feel that significant attention has not been given to it. We used simple Apriori-based techniques for feature extraction that runs quickly and gives good accuracies. Most of the previous work has used n-grams technique in order to retrieve it's features. We, however, saw that very good accuracies can be achieved without doing the expensive task of considering every possible combination of n-grams in a file. The intuition is that it is necessary to run this analysis quickly so that malware can be detected in time before the attack is launched. Hence, it is important to leverage the power of Machine Learning techniques in *acceptable* time so as to identify the malware before it's too late.

3 Data

Classification of malwares require utilization of high volumes of distinctive files, which is a consequence of authors introducing polymorphism and obfuscated code into their malwares. In order to build an effective model for analyzing and classification of these malwares our project uses a subset (10 GB) of the Microsoft Malware Classification [6] raw data set consisting of known malware files representing a mix of 9 different malware families.

Malware Family Table	
Class Label	Malware Family
1	Ramnit
2	Lollipop
3	Kelihos_ver3
4	Vundo
5	Simda
6	Tracur
7	Kelihos_ver1
8	ObfuscatorÅCY
9	Gatakp

Each unique malware is segregated on the basis of two features.

1. **ID #**: Made up of 20 character hash value, giving each file an unique identity.
2. **Class Label** : An integer ranging from $[1-9]$ representing the family name of the malware file.

Each malware further comprises of two types of file,

1. **Binary File**: Raw data representing the file's binary content (hex dump) in **hexadecimal format**.
2. **ASM File**: Meta-data manifest consisting of a log of the various meta-data information (binary, strings, function calls) obtained from the binary file using a commercial IDA disassembler.

The entire data-set was partitioned equally into training, validation and testing data-sets for the purpose of model building and testing. A sample representation of one of the malware files is shown below,

Sample Data Representation
Filename : 0A32eTdBKayjCWhZqDOQ **Class Label** : 2 - {Lollipop}

Snapshot Binary File - $0A32eTdBKayjCWhZqDOQ.bytes$

00401200	04	51	52	FF	D0	C7	05	AC	49	52	00	00	00
00401210	02	00	00	00	C2	08	00	CC	CC	CC	CC	CC	CC
00401220	6A	04	68	00	10	00	00	68	66	3E	18	00	6A

Snapshot ASM File - $0A32eTdBKayjCWhZqDOQ.asm$

.text:00401217	CC	CC	CC	CC	CC	CC	CC	CC	CC	CC	align	10h
.text:00401220	6A	04									push	4
.text:00401222	68	00	10	00	00						push	1000h

4 Methods

Following were the iterations that were executed to achieve the results given below:

4.1 Iteration 1: Modeling based on Feature Extraction

- We started with 10GB of data, and partitioned it into 3 different folders: training, validation and testing data-sets respectively.
- Next we extracted some basic features. Our basic feature set included a count of all the commonly occurring assembly language instructions.
- We then included some dynamic features which were derived from the files using feature-item-set techniques. The entire data was then normalized.
- Then we compared different classification techniques and plotted the accuracies [depicted in Fig 1.]. We observed that Random Forest gave the best accuracies.

4.2 Iteration 2: Modeling after Feature Selection and Parameter Tuning for Dimensionality Reduction

- The features so far used were raw features and there was no feature selection performed on it yet. In order to perform feature selection, dimensionality reduction was employed. It was discovered that application of PCA wasn't useful.
- We investigated the cause and found that our features showed no correlation and had counts which were non-negative. So instead we used a different technique, to get the desired result of omitting features whose presence in the entire data-set were rare. A dynamic threshold limit was estimated on the basis of the training data-set size and all features below this threshold were weeded out much like the confidence limits of Apriori algorithm.

- After the above feature selection, we ran the same classification algorithms that we did before, to see a comparison. We saw a huge increase in the accuracies [depicted in Fig 2.]. K-Nearest Neighbor saw a huge increase from 60% to 90%.

- Till now we were using the default parameters for classification, we then went ahead and used cross-validation in order to obtain the best parameters. This parameter tuning also resulted in an increase in the accuracies as shown in [depicted in Fig 3.].

4.3 Iteration 3: Interpret Best Feature in the Model Building Process

- We then evaluated the performance of our feature selection by checking how the accuracies are affected by eliminating one feature at a time. [depicted in Fig 4. and Fig 5.]

- A detailed analysis of our results and observations is summarized in the Analysis Section. 6.

5 Results

5.1 Iteration 1: Results

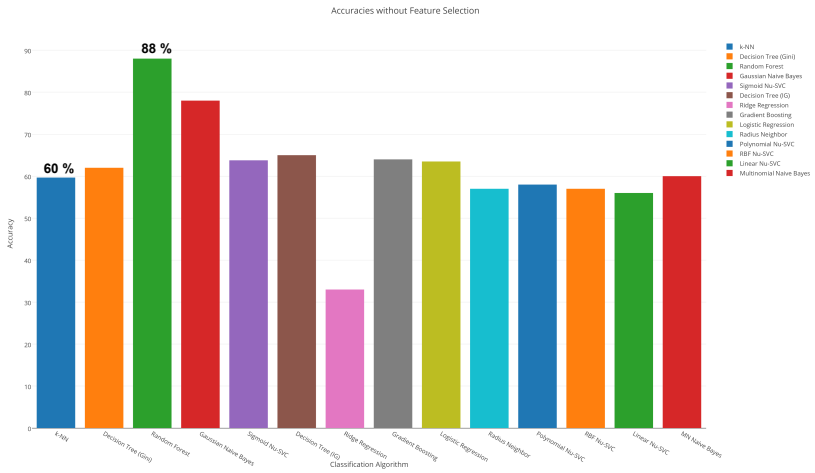

Fig 1. Accuracies without Feature Selection

5.2 Iteration 2: Results

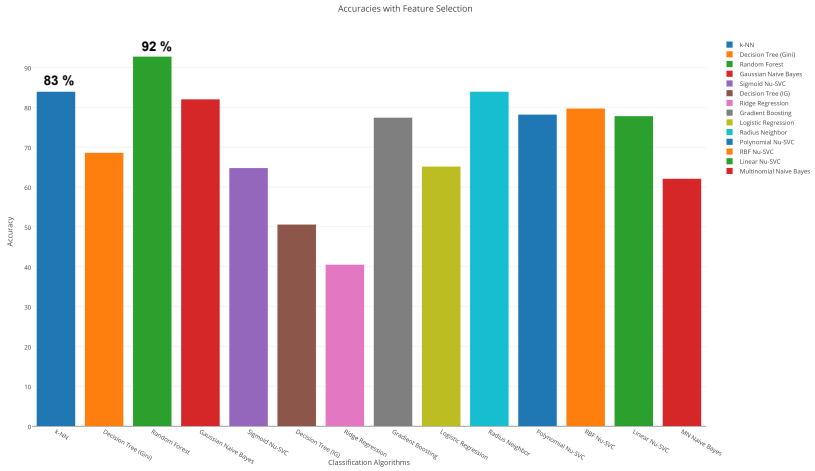

Fig 2. Accuracies with Feature Selection

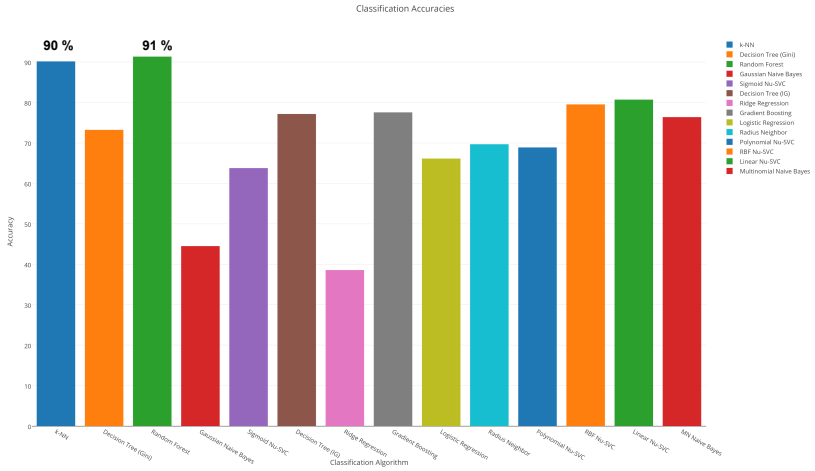

Fig 3. Accuracies with Feature Selection & Parameter Tuning

5.3 Iteration 3: Results

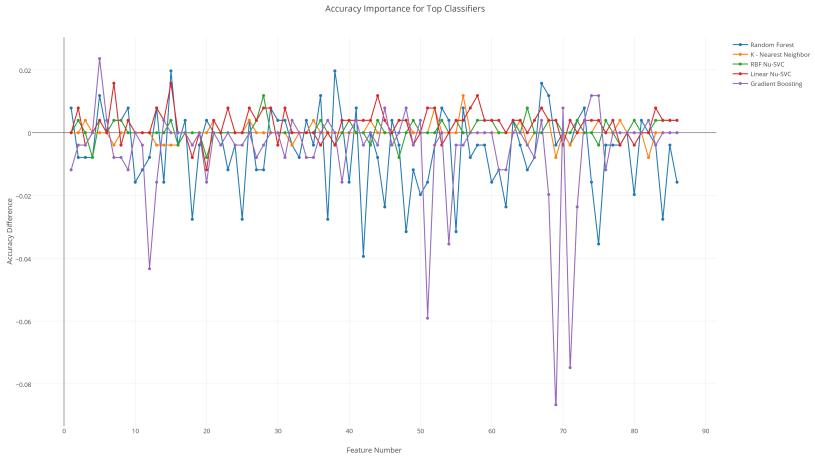

Fig 4. Accuracy Deviation - Top Classifier

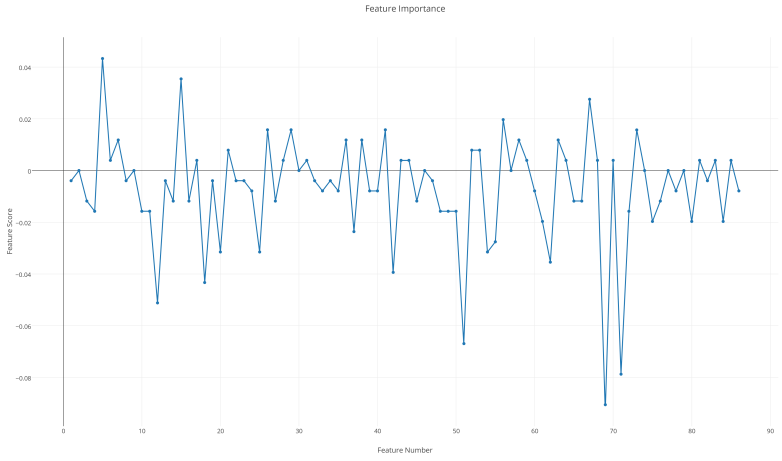

Fig 5. Accuracy Deviation - Cumulative

6 Analysis

6.1 Analysis of Iteration 1

- Our modeling uncovered that all our features could be segregated as either sparse or dense features.
- A large number of sparse features degraded the performance of most of the models since a lot of non-relevant features were being used for model construction.
- The innate randomness of curating dense nodes from subset of features gives Random Forest the robustness against over-fitting. Since the model is created using through dense features it gave the best performance even before feature selection.
- Apart from Random Forest all the other classifiers did not perform well before feature selection

6.2 Analysis of Iteration 2

The 2nd iteration of our modeling after performing feature selection and parameter tuning for dimensionality reduction [depicted in Fig 2.] can be summarized in the following table.

Analysis of Classification Accuracy		
Classification Algorithm	Accuracy	Analysis
K-Nearest Neighbor	Increase - 90 %	Redundant features which were causing distortion in the distance measure were removed hence construction of model with relevant features increased accuracy
Decision (Gini) / Decision (IG)	Inconsistent	Decision tree internal implementation shows that features are chosen at random and the 'best' possible split is found, according to some criterion. Due to this randomness a comparison between the accuracies before and after feature selection would be futile.
Random Forests	Increase - 92%	The exact feature set varies in each run but the performance remained stable throughout.
Gaussian Naive Bayes	Decrease - 45 %	Anomalous result which can't be attributed to the feature selection iteration.
Sigmoid Nu-SVC	Marginal Increase	No significant analysis
Logistic / Ridge Regression	Same	No significant analysis as regression is not suitable as per the nature of dataset.
Gradient Boosting	Increase - 78%	More accurate decision stumps created due to less number of sparse features leading to an overall increase in the ensemble model.
Radius Neighbor	Increase - 69%	After removal of noisy features the neighbors within the optimum radius for each data point will only comprise of relevant features.
Polynomial/RBF/Linear Nu-SVC	Increase - 80 %	High dimensional feature space had caused SVCs to suffer due to irrelevant features.
Multinomial Naive Bayes	Increase - 77 %	Since MNB works well with discrete features, removal of useless features gave an accurate model.

6.3 Analysis of Iteration 3

- The elimination of most features resulted in a significant decrease in accuracies [valleys depicted in Fig 4. & Fig 5.]
- The elimination of very few features (3) resulted in increase in accuracy (hardly by 0.05 %) which can be interpreted as noisy features.
- This analysis revealed that our feature selection resulted in 96.5 % of good features and the noisy features did not affect the accuracy significantly. ASM file instruction density features like **??, 00** were found to be the best features

7 Software Tools/Libraries

The entire project was written in **Python 2.7.9** and the libraries that were used include,

1. *scikit-learn 0.16.1* - data analysis and applying machine learning classification algorithms.
2. *numpy 1.9.2* - scientific computing and manipulation of feature vectors.
3. *scipy 0.15.1* - supplemented the numpy library for optimization, statistics.

8 Conclusion

We successfully classified the malwares into their respective families with a maximum accuracy of 91% achieved for Random Forest. We achieved these results with minimal overhead of the algorithms used to extract, select and classify the features. The methodology can be used to classify the malwares into their respective families, even if obfuscation and polymorphism has been employed to change the look and feel of the malware.

9 Future Work

Other classification techniques like Deep Learning can be used, which we couldn't use because of hardware limitations. However, it should be noted that techniques like deep learning can take a lot of time to run and so efficient methods must be thought of in order to ensure timely detection of malwares. There might still be tricks that can be employed to further reduce the speed of malware classification while still ensuring acceptable levels of accuracies.

Acknowledgments

We would like to express our deep gratitude and thank Prof. Ameet Talwalkar and Nikolas Karianakis for their support and assistance in this project and for providing us with the necessary guidance throughout the quarter.

References

[1] Asaf Shabtai, Robert Moskovitch, Clint Feher, Shlomi Dolev & Yuval Elovici (2012). Detecting unknown malicious code by applying classification techniques on OpCode patterns, *Springer-Verlag*, Vol 1. No. 1.

[2] GitHub : Microsoft Malware Kaggle
`https://github.com/xiaozhouwang/kaggle_Microsoft_Malware`

[3] Shabtai A., Moskovitch R., Elovici Y., Glezer C. (2009) Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Tech. Report* Volume 14 Issue 1, February, 2009 Pages 16-29.

[4] Muazzam A.S, Data Mining Methods for Malware Detection (2008), *M.S. University of Central Florida*, Doctoral Dissertation.

[5] Kolter, J. Zico and Maloof, Marcus A. (2006) Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research* Vol. 7, 2006 Pages 2721–2744.

[6] Kaggle : Microsoft Malware Challenge
`https://www.kaggle.com/c/malware-classification/data`