Detection of Malicious executable files using Machine Learning

1. Motivation

Malware detection using signature/heuristic-based methods is becoming more difficult since all current malware applications tend to use polymorphic layers or side mechanisms to update themselves to a newer version in a short period. Using the Microsoft malware dataset, we would like to explore the utility of different machine-learning techniques for the classification of executables into different types of malware classes.

2. Problem Statement

Malware is getting increasingly wider in complexity and reach, and despite this alarming fact, there is yet to be an effective method of quickly identifying malware and the potential effect it can have on our devices. We wish to provide a systematic study of applying machine learning techniques on Microsoft executables, while also incorporating domain knowledge to understand the importance of said features in identifying types of malware.

3. Related works

3.1. Malware detection using Linear SVM by Baigaltugs et al.

The study converts each executable into a vector using frequency counting and TFIDF methods. Vectors are sorted using a linear SVM, with different kernels being tested. The data consists of 40 different classes.

3.2. Malware Detection using Machine Learning by Dragos¸ Gavrilut et al.

The study extracts features from binary files and converts them into a binary vector with 308 features. The training is done to minimize the number of false positives, with the number of benign files being much greater than the number of malware files. Various perceptron algorithms like One-Sided Perceptron, Kernelized One-Sided Perceptron, and Cascade Classification are utilized for classification.

3.3. Malware Detection using Machine Learning and Deep Learning by Hemant Rathore et al.

The study uses opcode frequency as a feature vector, and a combination of unsupervised and supervised learning for binary classification of executables into malware and benign classes. Malware sources were downloaded from an open-source repository called the Malicia project, and regular Microsoft source files served as benign samples. Dimensionality reduction using auto-encoders and variance thresholds is used to reduce the complexity. The focus is on Random Forests and Deep learning-based models. The Random forest-based classifier is shown to outperform the deep learning-based classifier. ‘

4. Dataset Description

The dataset chosen is the Microsoft Malware Classification dataset, containing 10,868 samples. Link
Each sample contains a .byte file, which is a hexadecimal representation of the original executable, and a .asm file, which is the assembly dump obtained using the IDA tool, a name that is a 20-character-long unique hash value, and an integral label describing which class the malware belongs to, of which there are nine types. The assembly dump also contains a metadata dump.
Since the dataset does not provide direct features, we have tried several approaches to obtain relevant features.

4.1. Bag of Words

Each line in the byte file is of the form
<memory address> <series of 8-bit hexadecimal characters>
We have used each 256 hexadecimal character and ’??’ to create a bag-of-words vector for each sample.
Similarly, we have used the opcodes and assembly headers (.bss, .data, etc.) to create a bag-of-words vector for each sample. As seen later, some features are more important than others.

4.2. Feature generation using Domain Knowledge

We intend to use domain knowledge to generate features from each file. Some of these features (eg, file size, number of DLLs, etc.), which can be obtained trivially and is well known to be an indicator of malware files, have been extracted and used. Other features would require using nontrivial tools/scripts to generate and are left for future work.

4.3. Combining the above

We have combined the above two approaches to generate a comprehensive dataset containing 308 total features. 3 features (bss, data, and rtn) have a correlation of 1 with the other variables and have been dropped.

By using the above techniques, we obtain three datasets, one obtained from byte files, one obtained from asm files, and one which combines features of all 3. Due to the skewed nature of the dataset, oversampling is done to achieve class balancing, resulting in 26,478 samples. The data is split into training and testing sets following an 80-20 random split. The training set is further split using an 80-20 split to get a validation set.

5. Methodology and Models

For each of the three datasets obtained, the same methodology is followed to train five different models, Logistic Regression, Naive Bayes and Random Forests, Support Vector Machines, and Artificial Neural Networks. Individual steps carried out for each are described below.

5.1. Logistic Regression

Both original and oversampled datasets are utilized when training the models. Grid Search CV and Random Search CV are used to find the hyperparameters c (depth of regularization) and choice of loss functions (l1, l2, and elastic net).

5.2. Naive Bayes

Both original and oversampled datasets are considered.

5.3. Random Forest

Both original and oversampled datasets are utilized when training the models. Grid Search CV and Random Search CV are used to find the hyperparameters, number of estimators, max depth, and criterion (log-loss, gini, and entropy). We established a primitive feature ranking method using information gain obtained using RandomForest to make feature selection for bytes and combined datasets (this was not required for asm dataset, as it did not contain many features). The features with information gain less than a certain threshold were dropped. The performance obtained on the reduced feature set gave a lower generalization error.

5.4. SVM

Original and oversampled datasets were considered for all 3 types of files. For bytes files, certain features were dropped in order to give better performance

5.5. ANN

Both original and oversampled datasets for all 3 types of files are used. Keras models utilizing the GPU were used to speed up computation.

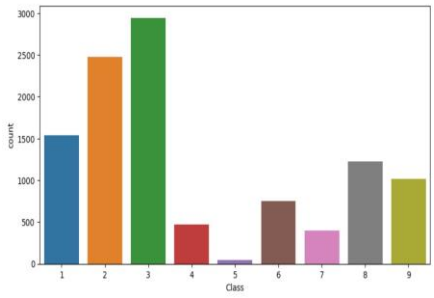

Figure 1. Class Distribution of the Dataset

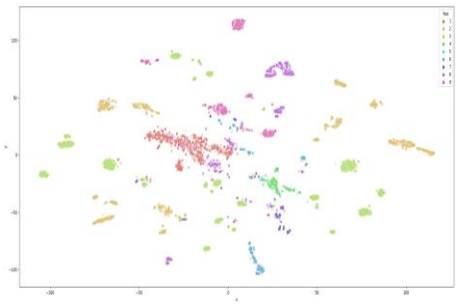

Figure 2. TSNE Plot for asm samples

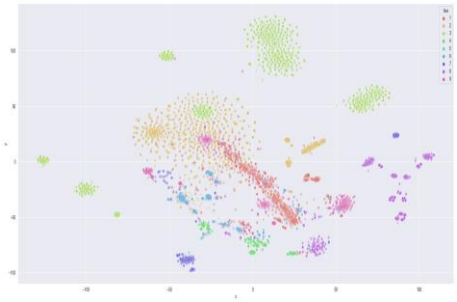

Figure 3. TSNE Plot for byte samples

6. Results and Analysis

We will use the log-loss of predicted probabilities and the accuracy score to compare different models. The performance trends observed are Logistic Regression < Gaussian Naive Bayes < Support Vector Machine < Artificial

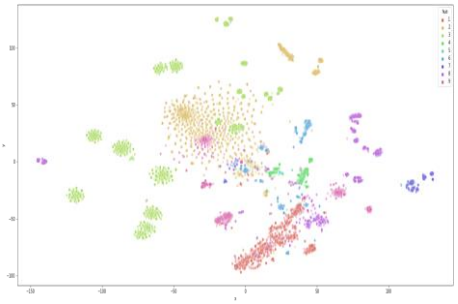

Figure 4. TSNE Plot for combined samples

Neural Network < Random forest in terms of accuracy for all 3 datasets (with one exception). While there is no way to definitively conclude between asm and bytes datasets in terms of performance, the combined feature set gives better results than the other two.

6.1. Logistic Regression

This is the worst-performing model of the five tested, with performance being poorer than Naive Bayes. It is also the model whose performance degrades the most with oversampling, except for the one trained on the combined dataset, where it shows improvement. When trained on the byte dataset, it gave an accuracy of < 0.5 for both original and oversampled datasets. There was no significant difference between the training and CV loss, indicating low variance.

6.2. Naive Bayes

Naive Bayes is the only model that gave poorer results in all oversampling cases, with much higher log losses being encountered for oversampled instances of all three types of datasets. As mentioned above, there was no noticeable difference in training and CV losses, indicating low variance.

6.3. Random Forest

Random forests perform the best among the models tested, with > 0.98 accuracy scores and consistently benefit from oversampled datasets. When training the model using all the features in the combined dataset, we encountered a scenario where the CV loss was much higher than the training loss, especially considering the low training error. In order to rectify this, we used the information gain given by the random forest classifier and ranked each feature based on it. By dropping features with information gain less than a threshold value, we obtained a new reduced feature set. Training a model on the reduced feature set results in the best performance of all the models tested.

During this process, we also tried to ascertain the difference in time taken and performance of the RandomCV and GridSearchCV modules for hyperparameter tuning.

GridSearchCV, due to checking all possible hyperparameters provided in a given range, tends to take of lot more computation time. In comparison, RandomSearchCV gives the best hyperparameter sampled from distributions for each hyperparameter tested. This makes it much faster to obtain a ”good enough” set of hyperparameters, especially in scenarios where quick deployment is essential. With the nature of malwares constantly evolving, quick deployment of models could become essential, especially in critical applications.

As expected, the hyperparameters provided by RandomSearchCV perform worse than the ones provided by GridSearchCV. However, with the difference in the accuracy being less than 0.5% in the case of RandomForest, the tradeoff regarding quicker training time can be helpful in specific scenarios.

6.4. SVM

The results of SVM classifier was just slightly lower than the results for RF, indicating the inherent inseparability of our dataset. Like RF, there is a slight increase in performance when using the oversampled dataset, and the performance trend between the three types of datasets follows Byte < ASM < Combined. The best-performing kernel was the RBF kernel.

6.5. ANN

As noted above, the performance of ANN lies between SVM and RF, except in the case of Byte features, where it was the worst-performing model by far. The ASM and combined features gave good performance, and all three types of datasets benefitted from oversampling. To avoid long computation times, we used the Keras implementation of ANN, with dropoff (p=0.5) and early stoppage. The activation function used was Leaky ReLU, and the optimizer used was Adam.

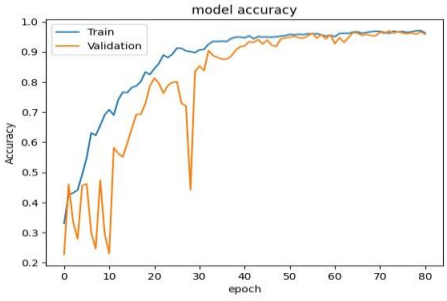

Figure 5. train-val loss curve for ASM dataset

7. Conclusions

We have shown that machine learning models based on simple features extracted from executables can be used to classify different types of malwares. We also tried to understand the effect of features on the variance of a machine-learning model. Finally, we have tried to analyze the benefits and tradeoffs of two hyperparameter testing frameworks.

To build on this work, in cases of availability of live executables, more quantitative information such as usage of the network stack, memory usage, number of calls to specific functions, files read, etc. can be obtained during runtime, which can significantly help in the quick analysis and deployment of models in order to be more proactive against malware detection and classification.

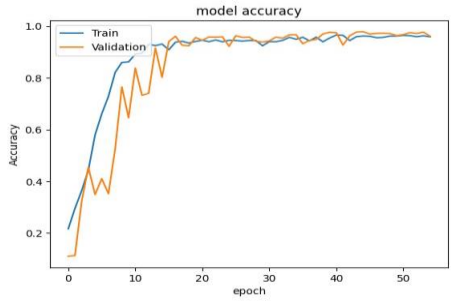

Figure 6. train-val loss curve for ASM oversampled dataset

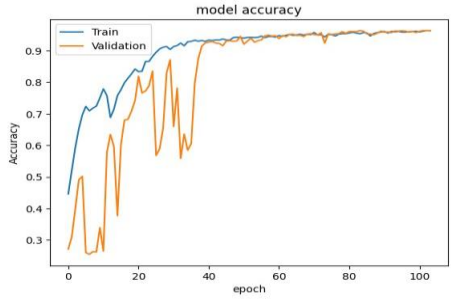

Figure 7. train-val loss curve for combined dataset

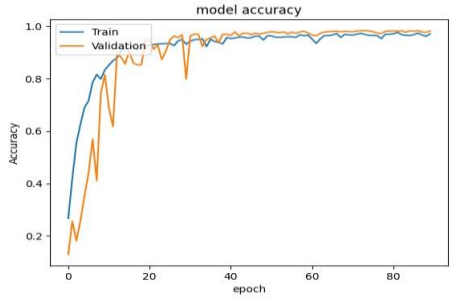

Figure 8. Train-val loss curve for combined oversampled dataset

8. References

1. D. Gavrilu¸t, M. Cimpoes¸u, D. Anton and L. Ciortuz,"Malware detection using machine learning," 2009 International Multiconference on Computer Science and Information Technology, 2009, pp. 735-741, doi: 10.1109/IMCSIT.2009.5352759.

2. Rathore, H., Agarwal, S., Sahay, S.K., Sewak, M. (2018). Malware Detection Using Machine Learning and Deep Learning. In: Mondal, A., Gupta, H., Srivastava, J., Reddy, P., Somayajulu, D. (eds) Big Data Analytics. BDA 2018. Lecture Notes in Computer Science(), vol 11297. Springer, Cham. https://doi.org/10.1007/978-3-030-04780-128

3. B. Sanjaa and E. Chuluun, "Malware detection using linear SVM," Ifost, 2013, pp. 136-138, doi: 10.1109/IFOST.2013.6616872.

4. Link to GitHub repository

Model	Original Data				Oversampled Data			
	Accuracy	Train Loss	CV loss	Test Loss	Accuracy	Train Loss	CV loss	Test Loss
ByteFile Features	0.6476	1.0937	1.1366	1.1521	0.64558	1.3227	1.3367	1.263
Asm Features	0.6872	1.0115	1.0127	0.9982	0.6297	1.2601	1.2767	1.2368
ByteFile + Asm	0.6987	0.9272	0.9234	0.9598	0.6334	1.206	1.2093	1.207

Table 1. Results for Naive Bayes

Table 2. Results for Random Search

Model	Original Data				Oversampled Data			
	Accuracy	Train Loss	CV loss	Test Loss	Accuracy	Train Loss	CV loss	Test Loss
ByteFile Features								
Logistic Regression	0.5896	1.1504	1.1482	1.1694	0.586	1.5779	1.5703	1.3901
Random Forest	0.9825	0.026	0.0641	0.078	0.9967	0.0191	0.0344	0.0267
Asm Features								
Logistic Regression	0.4144	1.683	1.6966	1.6846	0.2528	2.001	1.9925	1.9918
Random Forest	0.9898	0.026	0.0599	0.0551	0.9947	0.0094	0.021	0.0234
ByteFile + Asm								
Logistic Regression	0.6057	1.1765	1.1677	1.1704	0.6849	1.5337	1.5414	1.357
Random Forest	0.9912	0.0172	0.035	0.0413	0.9964	0.0082	0.0211	0.0199

Model		Original Data				Oversampled Data		
	Accuracy	Train Loss	CV loss	Test Loss	Accuracy	Train Loss	CV loss	Test Loss
ByteFile Features								
Logistic Regression	0.6007	1.1484	1.1634	1.1612	0.5188	1.1572	1.5629	1.4004
Random Forest	0.9806	0.0288	0.0783	0.082	0.994	0.0189	0.0428	0.0285
Asm Features								
Logistic Regression	0.6076	1.1364	1.1663	1.11	0.5381	1.5842	1.5922	1.3745
Random Forest	0.9912	0.0155	0.0487	0.0418	0.9945	0.0094	0.021	0.0233
ByteFile + Asm								
Logistic Regression	0.6251	1.1689	1.1672	1.1588	0.6881	1.5232	1.5397	1.3551
Random Forest	0.9908	0.0167	0.048	0.0426	0.9949	0.0088	0.0241	0.0218

Table 3. Results for Grid Search

Model	Original Data				Oversampled Data			
	Accuracy	Train Loss	CV loss	Test Loss	Accuracy	Train Loss	CV loss	Test Loss
ByteFile Features	0.94576	0.227	0.2811	0.2899	-	-	-	-
Asm Features	0.9577	0.277	0.3508	0.3733	0.972	0.1754	0.1978	0.2434
ByteFile + Asm	0.973	0.1543	0.2554	0.2071	0.979	0.0975	0.1354	0.1366

Table 4. Results for SVM

Model	Original Data			Oversampled Data		
	Train Acc	Val Acc	Test Acc	Train Acc	Val Acc	Test Acc
ByteFile Features	0.5169	0.5095	0.5092	0.9501	0.9452	0.9489
Asm Features	0.9733	0.9586	0.95998	0.9615	0.9608	0.9603
ByteFile + Asm	0.9760	0.9638	0.97148	0.9843	0.9809	0.9769

Table 5. Results for ANN