Malicious PE Detection using Random Forest Classification

1. Abstract:

Microsoft's Windows Operating System is one of the most widely used operating systems in the World. Windows was always able to attract users because of its application availability and a not so complicated interface. But when it came to security, Windows has been struggling historically. It is very likely that a large amount of malware designed to attack Windows operating system is in the Portable Executable (PE) file format. Static malware analysis does not require the malware to be run on the system, eliminating the possibility of any infection. Dynamic analysis is known to be the most efficient technique to identify malware and what it is intended to do. But once the malware realizes it is being run in a virtual environment, it might change its behavior and become undetectable. Using static analysis in combination with dynamic analysis can yield better results. Identifying malicious files before performing dynamic analysis can considerably reduce time and cost. This technique can be used as the first layer of security.

The accuracy of a machine learning model also depends on the feature set used to train the classifier. I have investigated the idea of using a Random Forest Classifier instead of the LightGBM model used in the original research. About 50,000 samples from the dataset provided in [1] were used to train the classifier on a select set of features. Machine learning models using Logistic Regression, Naïve Bayes, XGBoost and AdaBoost algorithms have been developed to compare the performance of the Random Forest classifier. Some of the features used in the original research such as sha256, byte histogram, and byte-entropy histogram. Our model was able to predict malicious PE file with a detection rate of 97.34% and AUC of 0.996 with default hyperparameters and detection rate of 98.6% and AUC of 0.998095 after hyperparameter tuning.

2. Related Work:

Many researchers have been investigating the use of machine learning algorithms to classify malware. The features used to determine the maliciousness of executables have changed from time to time. Some of the most commonly used features include using opcode, API calls, API call sequences, and PE32 header data.

Researchers have used opcode n-gram patterns to identify malicious files [2]. Executables were disassembled, and sequences of opcodes were extracted. These opcodes were later used by the classifier to train the model. The model was trained on features extracted from around 30,000 sample files with an accuracy of 96%.

The usability of Windows API calls to identify malware was widely investigated. [3] Uses the features derived from Windows API calls in the files to train the machine learning models rather than the Windows API call sequences. The dataset was prepared from 66,703 executable files. Models were trained with Naïve Bayes, KNN, Sequential Minimal Optimization (SMO), Backpropagation Neural Networks, J48 algorithms. Malicious files are generally crafted with different imports than the benign executables [4].

Roth and Henderson used byte histogram and byte entropy histogram with general PE32 static header information of over a million executables. The classifier was trained using Light Gradient Boosting Model (LightGBM) algorithm. The ROC AUC exceeds 0.99911 and a detection rate of 98.2% [1]. Researchers also used static PE32 header file features along with import functions. [5] was trained using features extracted from around 12,000 executables. Four different categories of malware were investigated. In [6], only five features were used to train the classifier on the features extracted from around 6700 executables. Bai, Wang, and Zou used 197 PE32 header features of around 20k executables and concluded that ensemble algorithms performed better.

Markel has investigated the presence of a class imbalance in malware training datasets. The dataset consists of features from 164,802 files. [7] used both static PE header files and also Windows API calls. Random Forest produced the best results at a malware prevalence of 0.5.

Jakub Acs [8] used only a set of import functions from the windows executables to train the data. The classifier was trained on features from 15000 PE files. Decision Trees algorithm turned out to be the most efficient model with FPR of 2.18%.SVM has been identified to be the most effective model to identify malware followed by J48, Naïve Bayes, and Random Forest Algorithm [9].

A different set of features have been studied to determine the features that would be most effective to differentiate between benign and malicious executables. Some studies used the static PE32 header data; Windows API calls, Opcode patterns while others used a combination of all of them. Other studies used only a certain number of features from the header to identify maliciousness in windows executables.

3. Problem Statement:

The goal is to develop a machine learning classifier to predict malicious PE files; the tasks involved are:

1. Preprocess the data
2. Train different classifiers that can predict the malicious PE files
3. Make the predictions on the test set
4. Evaluate the classifier using evaluation metrics

4. Introduction:

4.1 PE File Format:

The portable executable (PE) format is a file format used for executable, object code, DLLs, FON font files and others used in Windows OS. The PE file format is a structure that contains the information necessary for a Windows OS loader to manage the executable code. The file format includes many informational headers, followed by sections. Some of the PE filename extensions are .acm, .ax, .cpl, .dll, .drv, .efi, .exe, .mui, .ocx, .scr, .sys, .tsp . PE currently supports the IA-32, IA-64, x86-64, ARM and ARM64 instruction set architectures.

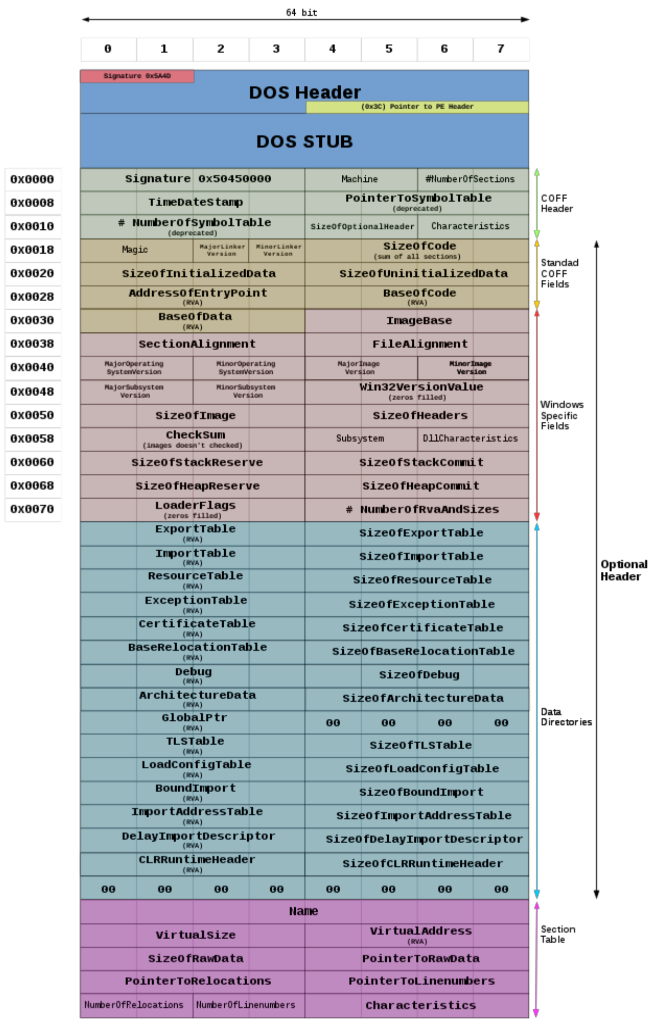

Fig 2: Structure of a 32-bit PE File

4.3 Random Forest Classifier:

Random Forest Algorithm is one of the most powerful and widely used machine learning algorithm. It makes use of feature bagging, which significantly decreases the correlation between each decision tree. Random Forests are an improvement over traditional bagged decision tree. The major difference between traditional decision trees and Random forests is that the process of finding the root node and splitting feature nodes run randomly. It grows many classification trees to classify a new object while each tree gives a classification. [10] The algorithm is built on the premise that building a small decision-tree with few features is a computationally cheap process. Random forests have a method to estimate missing data and can handle input without variable deletion.

There are two stages in the Random Forest algorithm.

1) Random Forest creation:
   For each tree in the forest, a bootstrap sample from S is selected. A modified decision tree algorithm is then used to learn a decision-tree. In traditional decision tree algorithms, at each node of the tree, all possible feature-splits are examined. [11] Instead of examining all possible feature splits, a select set of features '*f*' are randomly selected from the entire set of features "*F*". The node then splits on the best feature in "*f*" rather than "*F*".
2) Random Forest prediction:
   a) Take the features from the test set and use the rules of each randomly created decision tree to predict the target
   b) Calculate the votes for each predicted target
   c) Take the one with the highest voted predicted target as the final prediction.

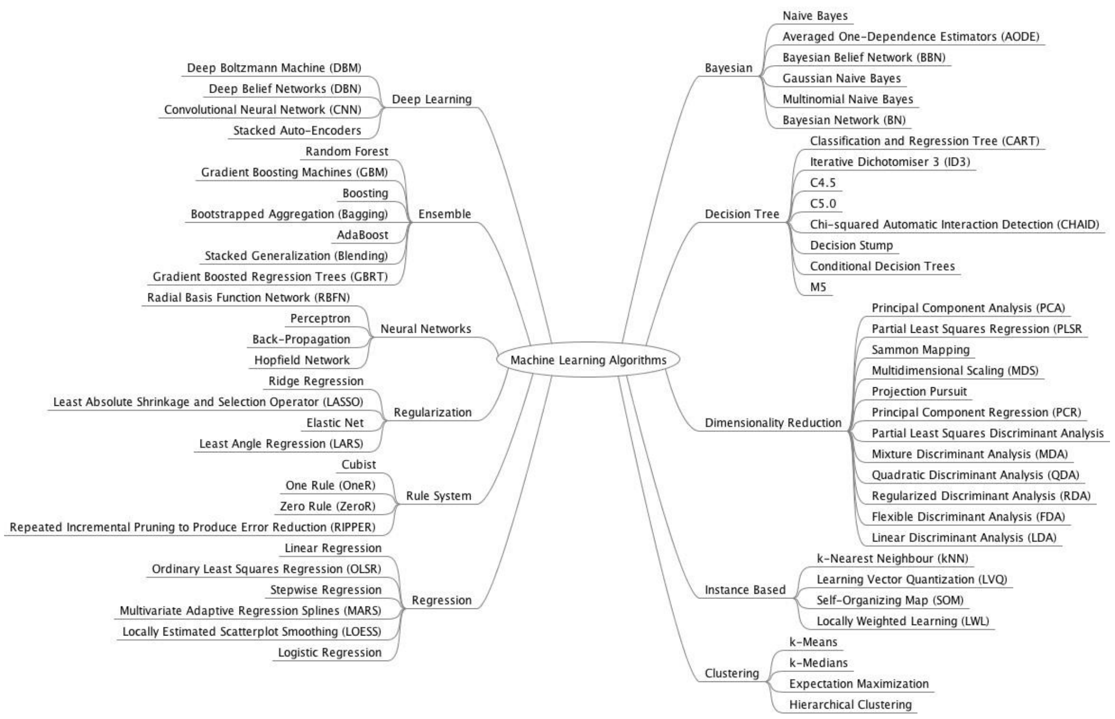
Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Fig 3: Some well-known machine learning algorithms.

5. Methodology:

5.1 Dataset Description:

The dataset includes features extracted from around 50,000 binary files as a collection of JSONlines, where each line contains a single JSON object. Each object consists of the PE header features shown below.

```
{'general': {'exports': 0,
  'has_debug': 0,
  'has_relocations': 1,
  'has_resources': 0,
  'has_signature': 0,
  'has_tls': 0,
  'imports': 41,
  'size': 33334,
  'symbols': 0,
  'vsize': 45056},
 'header':    {'coff':    {'machine':    'I386',
'timestamp': 1365446976},
  'optional': {'magic': 'PE32',
   'major_image_version': 1,
   'major_linker_version': 11,
   'major_operating_system_version': 6,
   'major_subsystem_version': 6,
   'minor_image_version': 2,
   'minor_linker_version': 0,
   'minor_operating_system_version': 0,
   'minor_subsystem_version': 0,
   'sizeof_code': 3584,
   'sizeof_headers': 1024,
   'sizeof_heap_commit': 4096,
   'subsystem': 'WINDOWS_CUI'}},
 'label': 1,
 'section': {'entry': '.text',
  'sections': [{'entropy': 6.368472139761825,
    'name': '.text',
    'size': 3584,
    'vsize': 3270},
   {'entropy': 7.924775969038312,
    'name': '.rdata',
    'size': 27136,
    'vsize': 26926},
   {'entropy':    0.9424221913195016,    'name':
'.data', 'size': 512, 'vsize': 528},
   {'entropy': 4.049286786417362,
    'name': '.reloc',
    'size': 1024,
    'vsize': 514}]},
 'strings': {'MZ': 1,
  'avlength': 8.170588235294117,
  'entropy': 6.259255409240723,
  'numstrings': 170,
  'paths': 0,
  'printables': 1389,
  'registry': 0,
  'urls': 0}}
```

The features in the dataset are later preprocessed in a way for the machine learning algorithm to train the model.

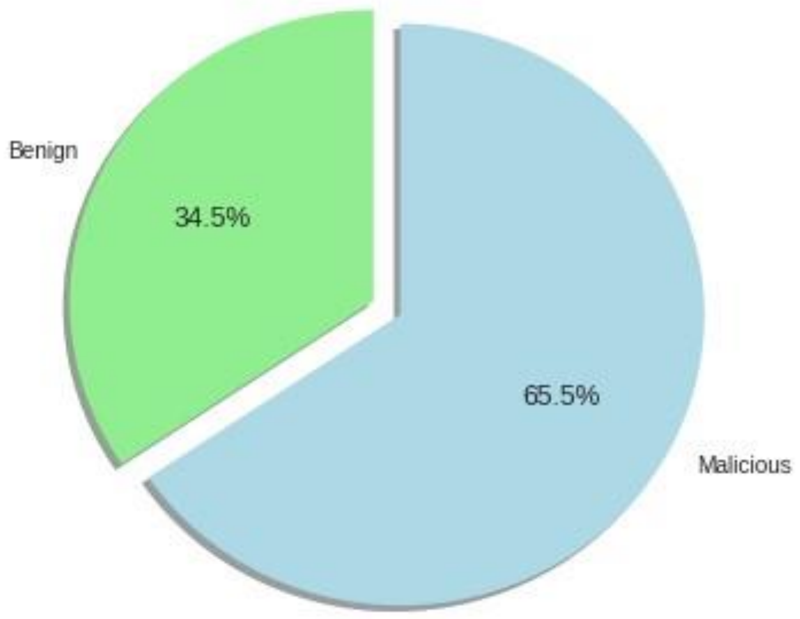

Fig 1: Class distribution of the dataset

5.2 Data Preprocessing:

The initial dataset had very complex JSON nesting, which cannot be directly put into a data frame. The dataset also has a fair number of categorical variables to be dealt with before the data can be used to build the model. The preprocessing phase consists of the following steps:

1. De-nest the JSON data
2. Remove undesired features that can slow the learning
4. Deal with categorical data
5. Prepare the data frame accordingly
6. Fill in missing data in the data frame
7. Vectorize the data frame

5.3 Performance Evaluation Metrics:

i) **Accuracy:** Accuracy is one common metric for evaluating classification models. It takes both true positives and true negatives with equal weight.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can be calculated in terms of positives and negatives.

$$\text{Accuracy} = \frac{TP + FN}{TP + TN + FP + FN}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives and FN =False Negatives. If the set of predicted labels for a sample match with the real set of labels, the accuracy score is 1.0, otherwise it is 0.0.

ii) Area Under the ROC Curve (AUROC): ROC curve is a plot between the true positive rate (TPR) on the y-axis and False positive rate (FPR) on the x-axis at various thresholds. It is also called the sensitivity vs. (1 – specificity) plot. The best prediction method would yield a point in the upper left corner at (0, 1) on the graph; also called perfect classification. A random guess would yield a point along a diagonal line. The diagonal divides the ROC space into two; points above the diagonal generally represent better classifications and below the diagonal represent bad results. AUROC curve is a performance measurement for classification problems at various threshold settings. A higher area under the curve means the model is better at predicting 0's as 0's and 1's as 1's.

Both accuracy and AUROC were used to evaluate our model.

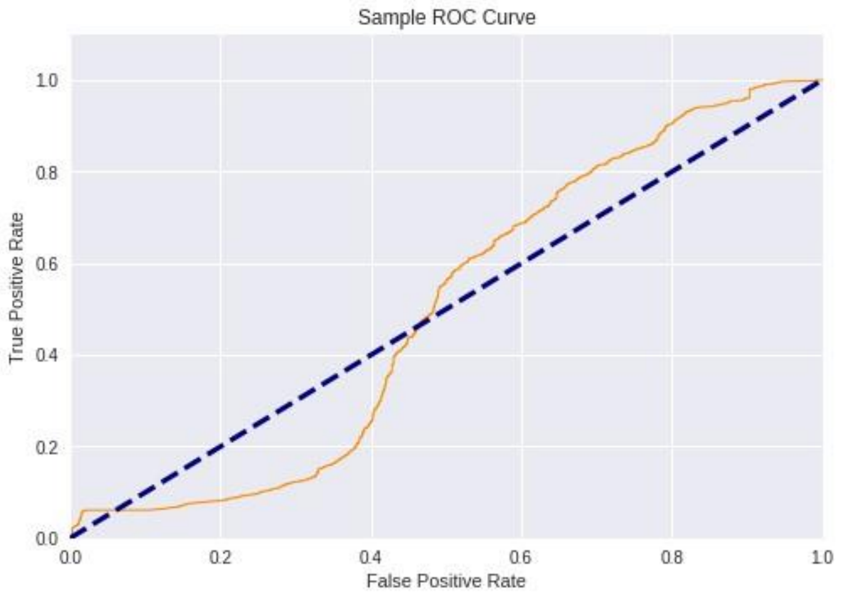

Fig 4: A Sample ROC Curve

5.4 Implementation:

This project is written in python and the implementation process can be split into two parts:

1. Download data and data preprocessing
2. Train the classifier
3. Evaluate the performance of the classifier

During the first stage, the dataset was downloaded, and the preprocessing was done using a collection of python packages [12] pandas, [13] numpy, flatten_json. The free cloud platform "Google Colaboratory” tool which is built on top of “Jupyter Notebook” was used. The dataset can be loaded into the Jupyter notebook by mounting on Google Drive.

To train the classifier, Random Forest Classification implementation provided by [14] sci-kit learn was employed. accuracy_score and roc_auc methods in sci-kit learn were used to evaluate the classifier.

6. Classifier Evaluation:

The Receiver Operating Characteristic Curve generated with different models that were developed is shown in Fig 6. It was possible to develop our model with the area under the curve exceeding 0.998, which is relatively higher than some of the most popularly used machine learning algorithms like Logistic Regression, Adaptive Boosting (AdaBoost), extreme Gradient Boosting (XGBoost) and Naïve Bayes.

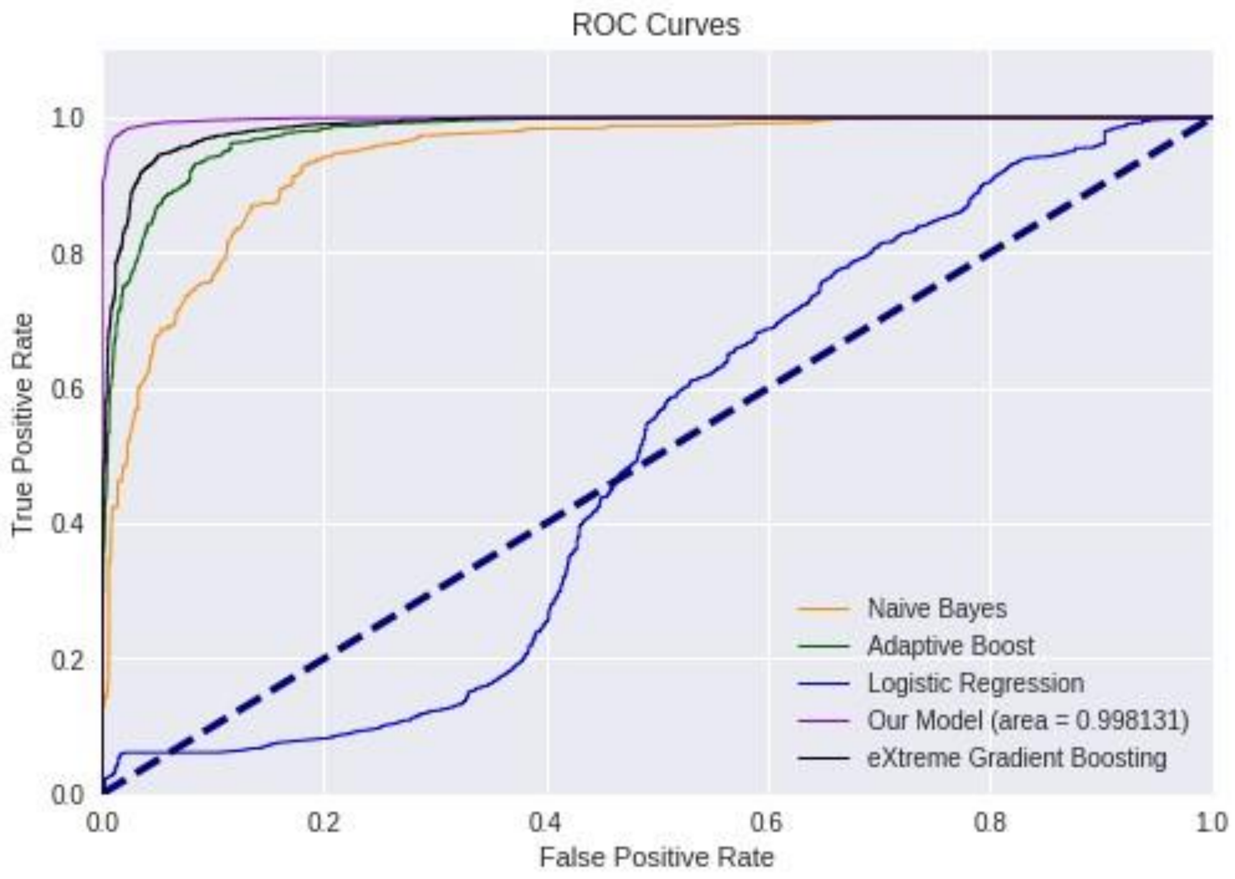

Fig 6: ROC Curves of our model against different algorithms.

The model predicts if the PE file is Benign or Malicious with an accuracy of 98.10% which is relatively higher than the accuracy of all the other models.

Model	Accuracy
Naïve Bayes	89.08%
Logistic Regression	66.28%
Adaptive Boosting(AdaBoost)	93.34%
Random Forest Classifier (Our Model)	98.10%
Extreme Gradient Boosting (XGBoost)	94.38%

Table 1: Accuracy of the developed models.

7. Discussion:

The use of Random Forest Algorithm for malicious PE detection is proposed using features from a dataset provided in [1]. The dataset is processed in order to generate a custom set of features that are used to train the classifier. Other classifiers like Naive Bayes, XGBoost, AdaBoost and Logistic Regression are developed to compare the results with the Random Forest Classifier. Accuracy and AUROC metrics were used to compare the classifiers developed. The performance of the proposed classification technique is the best among the classifiers developed. It is inferred that the use of Random Forest Classification techniques can yield better results when compared to those of Decision tree and Boosting techniques. The proposed model is on par with the LightGBM model used in the original research. Better feature selection techniques, hyperparameter tuning and continuously updating the dataset can improve the efficiency of the proposed model.

References

[1] Roth, P., & Anderson, H. S. (2018). EMBER: An Open Dataset for Training Static Malware Machine learning Models. ArXiv e-prints.

[2] Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on OpCode patterns. Security Informatics.

[3] Alazab, M., Sitalakshmi, V., Watters, P., & Alazab, M. (2011). Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures. Proceedings of the 9th Australasian Data Mining Conference (pp. 171-182). Ballarat, Australia: Australian Computer Society.

[4] Belaoued, M., & Mazouzi, S. (2014). Statistical Study of Imported APIs by PE Type Malware. International Conference on Advanced Networking Distributed Systems and Applications. Bejaia, Algeria.

[5] Vyas, R., Luo, X., McFarland, N., & Justice, C. (2017). Investigation of malicious portable executable file detection on network using supervised learning techniques. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management, (pp. 941-946). Lisbon, Portugal.

[6] Liao, Y. (n.d.). PE header-based Malware study and detection.

[7] Markel, A. Z. (2015). Machine Learning Based Malware Detection. Annapolis, Maryland: USNA Trident Scholar Project Report.

[8] Acs, J. (2018). Static detection of malicious PE files. Prague.

[9] Souri, A., & Hosseini, R. (2018). A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Computing and Information Sciences.

[10] Liaw, A., & Wiener, M. (2002, December). Classification and Regression by randomForest. R News, pp. 18-22.

[11] Random Forests. (n.d.). Retrieved from http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf.

[12] Machine Learning Algorithms Mindmap. (n.d.). Retrieved from https://jixta.wordpress.com/2015/07/17/machine-learning-algorithms-mindmap/

[13] Portable Executable. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Portable_Executable

[14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research 12, 2825-2830.

[15] Bai, J., Wang, J., & Guozhong, Z. (2014). A Malware Detection Scheme Based on Mining Format Information. Scientific World Journal.

[16] McKinney, W. (2011). pandas: a Foundational Python Library for Data Analysis and Statistics.

[17] Johann van der Walt, S., Chris Colbert, S., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science and Engineering 13, 22-30.

[18] Bradley, A. P. (1997). The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. Pattern Recognition, 1145-1159.

[19] Hossin, M., & M.N, S. (2015). A Review on Evaluation Metrics for Data Classification Evaluations.