Detecting Malware in Portable Executable files

Semester: 06

MINOR PROJECT 2
SUBJECT CODE:15B19CI691

Under the supervision of
Dr. Satish Chandra

Department of Computer Science and Engineering

Jaypee Institute of Information Technology
(Declared Deemed to be University U/S 3 of UGC Act)
A-10, SECTOR-62, NOIDA, INDIA

May 2021

ACKNOWLEDGEMENT

First of all, we would like to express our gratitude towards the Almighty for enabling us to complete this report on “Detecting Malware in Portable Executable Files”.

Successful completion of any project requires help from a number of people.So we would like to thank our parents and friends for being our constant support during the course of the project.

Also, most importantly, we would like to extend our sincere thanks towards our Project Supervisor Dr. Satish Chandra. Without his kind direction and proper guidance, this study would not have been a success. In every phase of this project, his supervision and guidance shaped this report to be completed successfully. Working on this project has helped us immensely in increasing our knowledge and skills.

The details of the group members are as follows:

NAME	ENROLLMENT NO.	BATCH	Personal Email ID	College Email Id
Rahul Singhal	18103316	B10	rahulsinghal1904@gmail.com	18103316@mail.jiit.ac.in
Himanshu Joshi	18103132	B4	himanshujiit2018@gmail.com	18103132@mail.jiit.ac.in
Pranjal Chandel	18103139	B4	pchandel2000@gmail.com	18103139@mail.jiit.ac.in

Introduction

Malware is an application that is harmful to your computer. Malware is a collection of a number of malicious software variants, including viruses, ransomware, and spyware. In this project, we are going to distinguish between malware and legitimate .exe files by looking at the properties of the Portable Executable files.
The Portable Executable format is a file format for executables, object code, DLLs, and others used in 32-bit and 64-bit versions of Windows operating systems. It can be difficult to effectively apply domain knowledge to the task of malware detection, which we define as building a classifier that indicates whether an executable binary is benign or malicious.
The format and nature of executable files are complicated and not always consistent. Just parsing a Portable Executable (PE) file correctly is difficult. We are going to use the header field in files to obtain features for our project. Then we will apply different machine learning models to differentiate malware files from non-malware ones and compare them on the basis of different performance metrics.

Objective

To detect and identify malware in portable executable(PE) files using different machine learning models and then compare them on the basis of accuracy and features.

Data Source and its Storage

We are have formulated the data set from the following sources:

https://archive.ics.uci.edu/ml/datasets/Detect+Malacious+Executable(AntiVirus)

https://marcoramilli.blogspot.com/2016/12/malware-training-sets-machine-learning.html

https://github.com/jivoi/awesome-ml-for-cybersecurity#-datasets

https://zeltser.com/malware-sample-sources/

Irrelevant features have negative effects on machine learning. Removing them improves the performance of algorithms, speeds them up, represents the problem better, and focuses the user’s attention on important variables.

We recognized that some features are correlated highly to each other and that these would be in the same parts of the PE file. In order to separate highly correlated features, we sorted the features in decreasing order of individual accuracy in classification and then categorized them into seven buckets according to where in the PE file the features originated. We would define a new bucket whenever we encountered a feature going down this list from a different part of the PE file.
The seven resulting buckets can be labeled DataDirectory, OptionalHeader, Imports, Exports, Resources, Sections, and FileHeader.

These are the features with the highest individual accuracy taken from each of the seven buckets described before. We attempt to explain why these seven features lend themselves to good classification from the definitions of these fields and inspecting our data afterward:

1) DebugSize. Denotes the size of the debug-directory table. Usually, Microsoft-related executable files have a debug directory. Hence many clean programs may have a non-zero value for DebugSize.

2) ImageVersion. Denotes the version of the file. It is user-definable and not related to the function of the program. Many clean programs have more versions and a larger image-version set. Most malware has an ImageVersion value of 0.

3) IatRVA. Denotes the relative-virtual address of the import-address table. The value of this feature is 4096 for most clean files and 0 or a very large value for virus files. Many malware may not use import functions or might obfuscate their import tables.

4) ExportSize. Denotes the size of the export table. Usually, only DLLs, not executable programs, have export tables. Hence the value of this feature may be non-zero for clean files, which contain many DLLs, and 0 for virus files.

5) resources. Denotes the size of the resource section. Some virus files may have no resources. Clean files may have larger resources.

6) VirtualSize2. Denotes the size of the second section. Many viruses have only one section and the value of this field is 0 for them.

7) NumberOfSections. Denotes the number of sections. The value of this feature varies in both virus and clean files and it is not clear from inspection how this feature helps separate malware and clean files.

Features

We have extracted the following fields from the PE File header:

S.No	Feature	Description
1	Major Image Version	Used to indicate the major version number of the application; in Microsoft Excel version 4.0, it would be 4.
2	Virtual Address	A Virtual Address (VA) is a linear address in the Virtual Address Space (VAS) of the current program. These addresses are 32 bit linear addresses.
3	Size of the IMAGE_DATA_DIRECTORY	This size is different depending on whether it's a 32 or 64-bit file. For 32-bit PE files, this field is usually 224. For 64-bit PE32+ files, it's usually 240.
4	OS Version	This gives the version of the operating system with which it is compatible.
5	Import Address	Import Address Table (IAT) is an array of several function pointers where the address of the imported function is written by Windows loader.
6	Table Address	An array of records, pointed to by tables, that describe the actual size and location of the resource data. These records are the leaves in the resource-description tree.
7	Resources Size	Denotes the size of the resource section. Some virus files may have no resources. Clean files may have larger resources.
8	Number Of Sections	This indicates the size of the section table, which immediately follows the headers.
9	Linker Version	A linker or link editor is a computer system program that takes one or more object files and combines them into a single executable file, so this gives the version of the same.
10	Size of Stack Reserve	The size of the stack to reserve. Only SizeOfStackCommit is committed; the rest is made available one page at a time until the reserve size is reached.

11	DLL Characteristics	Image file is a dynamic-link library (DLL).They contain additional information that is required by the linker and loader in Windows. The preferred address of the first byte of image when loaded into memory; must be a multiple of 64 K.
12	Export Table Size and Address	An array of RVAs of exported symbols. These are the actual addresses of the exported functions and data within the executable code and data sections. Other image files can import a symbol by using an index to this table by using the public name that corresponds to the ordinal if a public name is defined.
13	Number Of Import DLL	The import library only contains code to load the DLL and to implement calls to functions in the DLL. This contains the no. of imported DLLs.
14	Image Base	It is the address in virtual memory where the executable should be loaded to avoid any adjustment of absolute jump instructions in the code.
15	Number Of Import Functions	Import tables are made available with lowercase names, to differentiate them from the upper case basic structure names.
16	Number Of Symbols	The number of entries in the symbol table. This data can be used to locate the string table, which immediately follows the symbol table. This
17	DebugSize	Denotes the size of the debug-directory table. Usually, Microsoft-related executable files have a debug directory. Hence many clean programs may have a non-zero value for DebugSize.
18	DebugRVA	DebugRVA (debug relative virtual address) An RVA in the portable executable (PE) header, which has a value of zero, indicates the field has not used all tables, and structure fields must be united
19	ImageVersion	An image version is what is used to create a VM when using a Shared Image Gallery When you create a VM, the image version is used to create new disks for the VM
20	IatRVA	This holds the RVA of the Import Address Table (IAT). The structure and content of the import address table are identical to those of the Import Lookup Table until the file is bound.

21	ExportRVA	ExportRVA (export relative virtual address) RVA (relative virtual address) exports ordinals for table entry.
22	ExportNameLen	ExportNameLen sorted the features in decreasing order of individual accuracy in classification
23	ExportFunctionsCount	The export data section, named .edata, contains information about symbols that other images can access through dynamic linking.This gives the count of export functions used.
24	DLL name and Imported Symbols	Import functions are done using the internal DLL table, which contains all the available import functions.
25	Filename	The Name field in IMAGE_EXPORT_DIRECTORY contains the internal name of the module (i.e. original name that was used while building the module).

Information about Different Models Used

Gaussian Naive Bayes

Naive Bayes is a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique but has high functionality. They find use when the dimensionality of the inputs is high. When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. The likelihood of the features is assumed to be-

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Sometimes assume variance:

- is independent of Y (i.e., σi),
- or independent of Xi (i.e., σk)
- or both (i.e., σ)

Gaussian Naive Bayes supports continuous-valued features and models each as conforming to a Gaussian (normal) distribution.
An approach to creating a simple model is to assume that the data is described by a Gaussian distribution with no co-variance (independent dimensions) between dimensions. This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all that is needed to define such a distribution.

In this project, the features have been taken from the merged file of clean and malware datasets, and they are used as the input attributes to the model.

Random Forest

Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.
Random forest is a supervised learning algorithm.
Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.
One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at the random forest in classification since classification is sometimes considered the building block of machine learning.

Random forest adds additional randomness to the model while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

In this project, the features have been taken from the merged file of clean and malware datasets, and they are used as the input attributes to the model.

Logistic Regression

Logistic regression is one of the most fundamental and widely used Machine Learning Algorithms. Logistic regression is usually among the first few topics which people pick while learning predictive modeling. Logistic regression is not a regression algorithm but a probabilistic classification model. Classification in Machine Learning is a technique of learning, where an instance is mapped to one of many labels. The machine learns patterns from data in such a way that the learned representation successfully maps the original dimension to the suggested label/class without any intervention from a human expert.

Logistic regression has a sigmoidal curve. Multiclass classification with logistic regression can be done either through the one-vs-rest scheme in which for each class a binary classification problem of data belonging or not to that class is done, or changing the loss function to cross-entropy loss. In the multi-class logistic regression python Logistic Regression class, multi-class classification can be enabled/disabled by passing values to the argument called ‘‘multi_class’ in the constructor of the algorithm. By default, multi_class is set to ’ovr’.

In this project, the features have been taken from the merged file of clean and malware datasets and they are used as the input attributes to the model.

Linear Discriminant Analysis

Linear discriminant analysis (LDA) is generally used to classify patterns between two classes; however, it can be extended to classify multiple patterns. LDA assumes that all classes are linearly separable and according to this multiple linear discrimination functions representing several hyperplanes in the feature space are created to distinguish the classes. If there are two classes then the LDA draws one hyperplane and projects the data onto this hyperplane in such a way as to maximize the separation of the two categories. This hyperplane is created according to the two criteria considered simultaneously:

•Maximizing the distance between the means of two classes;

•Minimizing the variation between each category.

It provides accepted accuracy and is widely used in BCI systems.
Linear discriminant analysis (LDA) is a discriminant approach that attempts to model differences among samples assigned to certain groups. The aim of the method is to maximize the ratio of the between-group variance and the within-group variance. When the value of this ratio is at its maximum, then the samples within each group have the smallest possible scatter, and the groups are separated from one another the most.

In this project, the features have been taken from the merged file of clean and malware datasets, and they are used as the input attributes to the model.

K Neighbors Classifier

The K in the name of this classifier represents the k nearest neighbors, where k is an integer value specified by the user. Hence as the name suggests, this classifier implements learning based on the k nearest neighbors. The choice of the value of k is dependent on data.
k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists of giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.
A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.

In this project, the features have been taken from the merged file of clean and malware datasets, and they are used as the input attributes to the model.

Decision Tree Classifier

A Decision Tree is a predictive model expressed as a recursive partition of the feature space to subspaces that constitute a basis for prediction. A Decision Tree is a rooted directed tree. In DTs, nodes with outgoing edges are the internal nodes. All other nodes are terminal nodes or leaves of the DT. DTs classify using a set of hierarchical decisions on the features. The decisions made at internal nodes are the split criterion. In DTs, each leaf is assigned to one class or its probability. Small variations in the training set result in different splits leading to a different DT. Thus, the error contribution due to variance is large for DTs. Ensemble learning, discussed in the next section, can help palliate the error due to variance. Decision trees are one of the most commonly used predictive modeling algorithms in practice. The reasons for this are many. Some of the distinct advantages of using decision trees in many classification and prediction applications are explained below along with some common pitfalls.

1. Easy to interpret and explain to non-technical users
2. As we have seen in the few examples discussed so far, decision trees are very intuitive and easy to explain to non-technical people, who are typically the consumers of analytics.
3. Decision trees require relatively little effort from users for data preparation
4. If we have a data set consisting of widely ranging attributes, for example, revenues recorded in millions and loan age recorded in years, many algorithms require scale normalization before model building and application. Such variable transformations are not required with decision trees because the tree structure will remain the same with or without the transformation.
5. When we fit a decision tree to a training data, the top few nodes on which the tree is divided are the most essential variables within the data set and feature selection is completed automatically.

   In this project, the features have been taken from the merged file of clean and malware datasets and they are used as the input attributes to the model.

Support Vector Machines

The objective of Linear SVM is to find a hyperplane in an n-dimensional space that separates the data points to their potential classes. The hyperplane should be positioned with the maximum distance to the data points. The data points with the minimum distance to the hyperplane are called Support Vectors. Due to their close position, their_influence on the exact position of the hyperplane is bigger than other data points. It is a supervised machine learning algorithm that helps in classification or regression problems. It aims to find an optimal boundary between the possible outputs.

Simply put, SVM does complex data transformations depending on the selected kernel function, and based on that transformations, it tries to maximize the separation boundaries between your data points depending on the labels or classes you've defined. In its most simple type, SVM doesn't support multiclass classification natively. For multiclass classification, the same principle is utilized after breaking down the multi-classification problem into multiple binary classification problems.

The idea is to map data points to high dimensional space to gain mutual linear separation between every two classes. This is called a One-to-One approach, which breaks down the multiclass problem into multiple binary classification problems. A binary classifier per each pair of classes. Another approach one can use is One-to-Rest. In that approach, the breakdown is set to a binary classifier per class.

In this project, the features have been taken from the merged file of clean and malware datasets, and they are used as the input attributes to the model.

Steps involved during the project

- First of all, we will import all the needed libraries required in the project.
- To make our code more organized we start by creating a class that represents the PE File information as one object. We are using the python module pefile which is a multi-platform Python module to parse and work with Portable Executable (aka PE) files.
- Now we move on to write a small method that constructs a dictionary for each PE File. Thus each sample will be represented as a python dictionary where keys are the features and values are the value of each parsed field.

- pe2vec() method and PEFile class: In this, we have displayed the dataset for the Malware and Clean Samples.
- Then we extracted the PE file data, malware PE information.
- In this way, we loop through all samples in the folder and process each one of them, and then dump all those dictionaries into a CSV file that we will use later.
- Now we exported the dataset as a CSV file.
- We checked the correlation for each column in the data frame.
- Then we obtained the plots for malware correlation and clean correlation.
- Then we cleaned the correlation to obtain new values.
- Then we prepared the data for analysis using supervised learning.
- Then we merged the malware and clean datasets.
- We reviewed the merged and cleaned data.
- Then we displayed the distribution of class and held out part of the available data as test data.
- Now, we split the data to 70% for training and 30% for testing.
- We applied different machine learning algorithms, displayed their Training and Testing accuracy, and used the Confusion Matrix and Classification Report for each model as performance metrics.
- Finally, we evaluated all the models and displayed their accuracy to check which one of them is giving a comparatively good result. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

Performance Metric Used

We used the Confusion Matrix and Classification Report for measuring performance.

- Confusion Matrix-

| TN | FP |

| FN | TP |

- Recall is the True Positive rate and indicates the probability of a true result.

Recall = TP / (TP +FN) (How well the model is predicting true Malware)

- Precision = TP / (TP + FP) (Positive predictive value)

True positives (TP): The amount of labels, which were correctly identified by the classifier, that is assigned point labeled A to class A.

True negatives (TN): The amount of labels, which were correctly rejected by the classifier, that is assigned point labeled A' to class A'.

False positives (FP): The amount of labels, which were incorrectly identified by the classifier, that is assigned point labeled A' to class A.

False negatives (FN): The amount of labels, which were incorrectly rejected by the classifier, that is assigned point labeled A to class A'.

- Accuracy: Accuracy measures a fraction of the classifier's predictions that are correct, that is the number of correct assessments divided by the number of all assessments – (TN + TP)/(TN + TP + FN + FP).

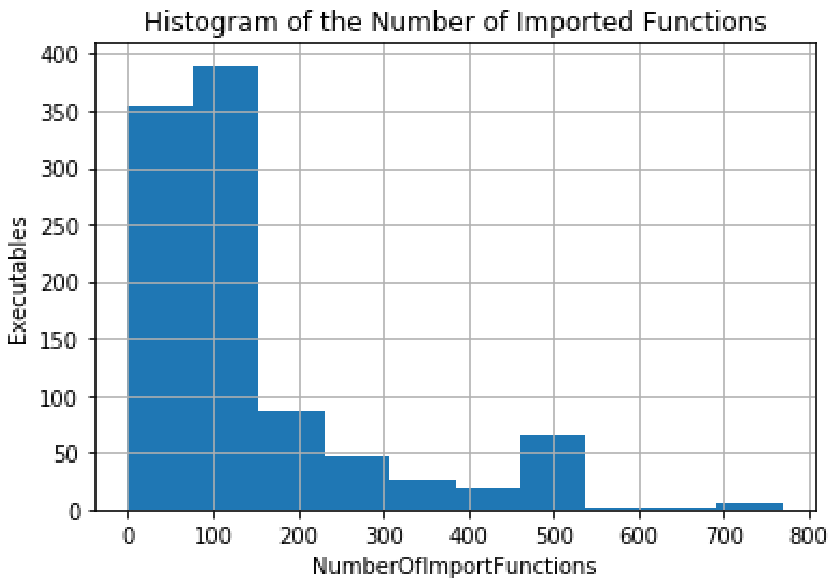
- Precision (P): Precision is the fraction of positive predictions that are correct – TP/(TP + FP). Be careful as a classifier predicting only a single positive instance, that happens to be correct, will achieve perfect precision.

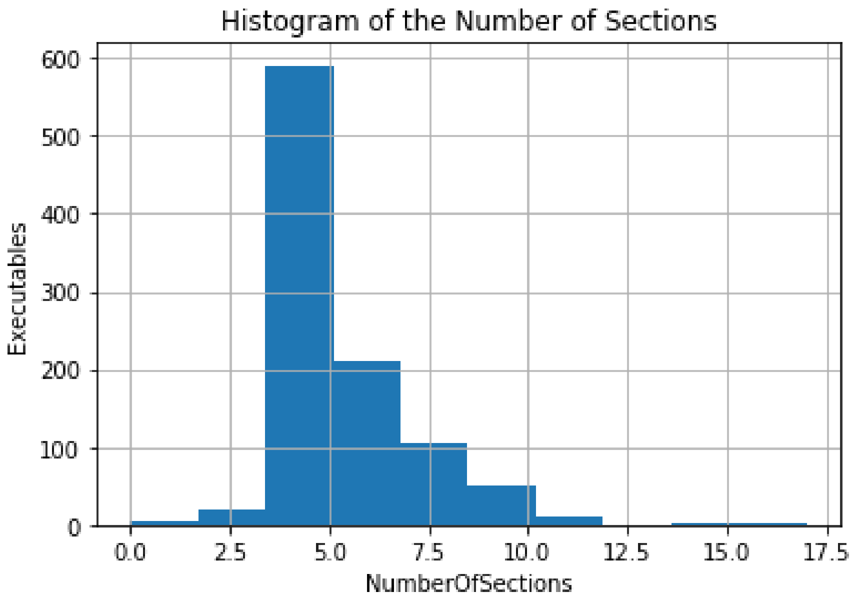
- Recall (R) Recall, sometimes called sensitivity in medical domains, measures the fraction of the truly positive instances. A score of 1 indicates that no false negatives were present – TP/(TP + FN). Be careful as a classifier predicting positive for every example will achieve a recall of 1.

- F1 score: Both precision and recall scores provide an incomplete view on the classifier performance and sometimes may provide skewed results. The F1 measure provides a better view by calculating weighted average of the scores – 2*P*R/(P + R). A model with perfect precision and recall scores will achieve an F1 score of one.

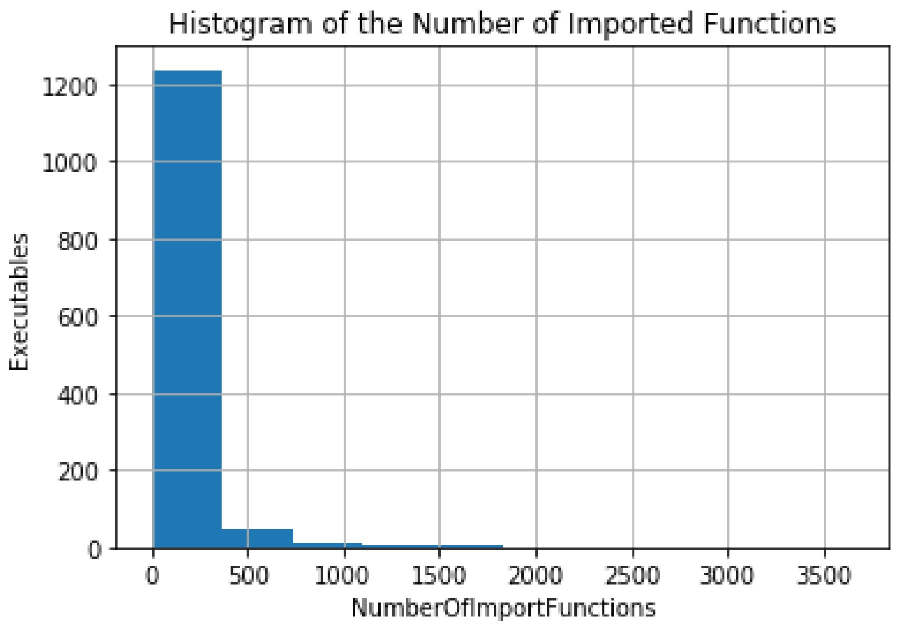
Observations derived during the course of the project

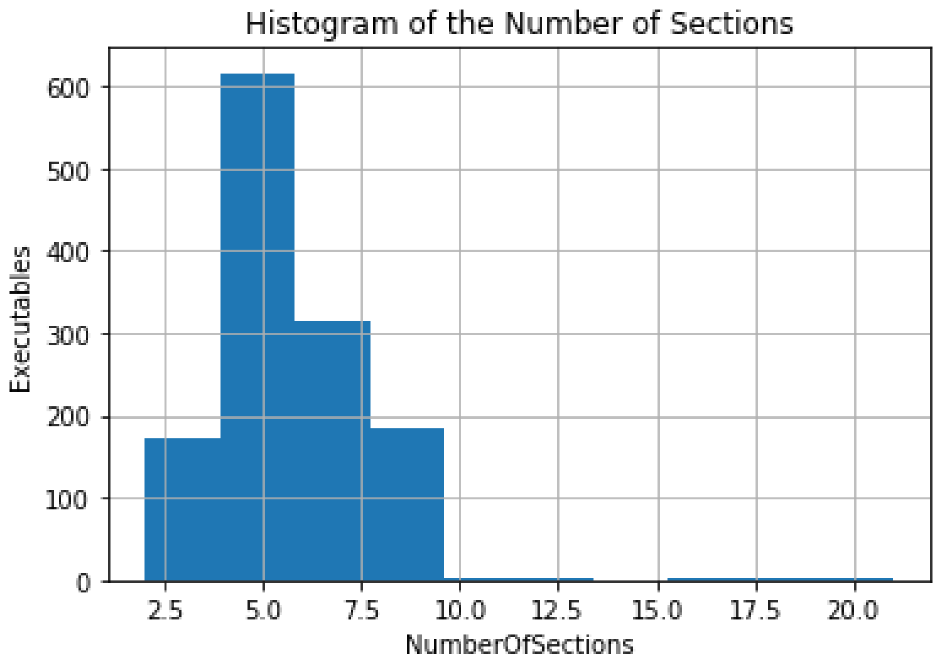
Histograms of Malware Dataframes



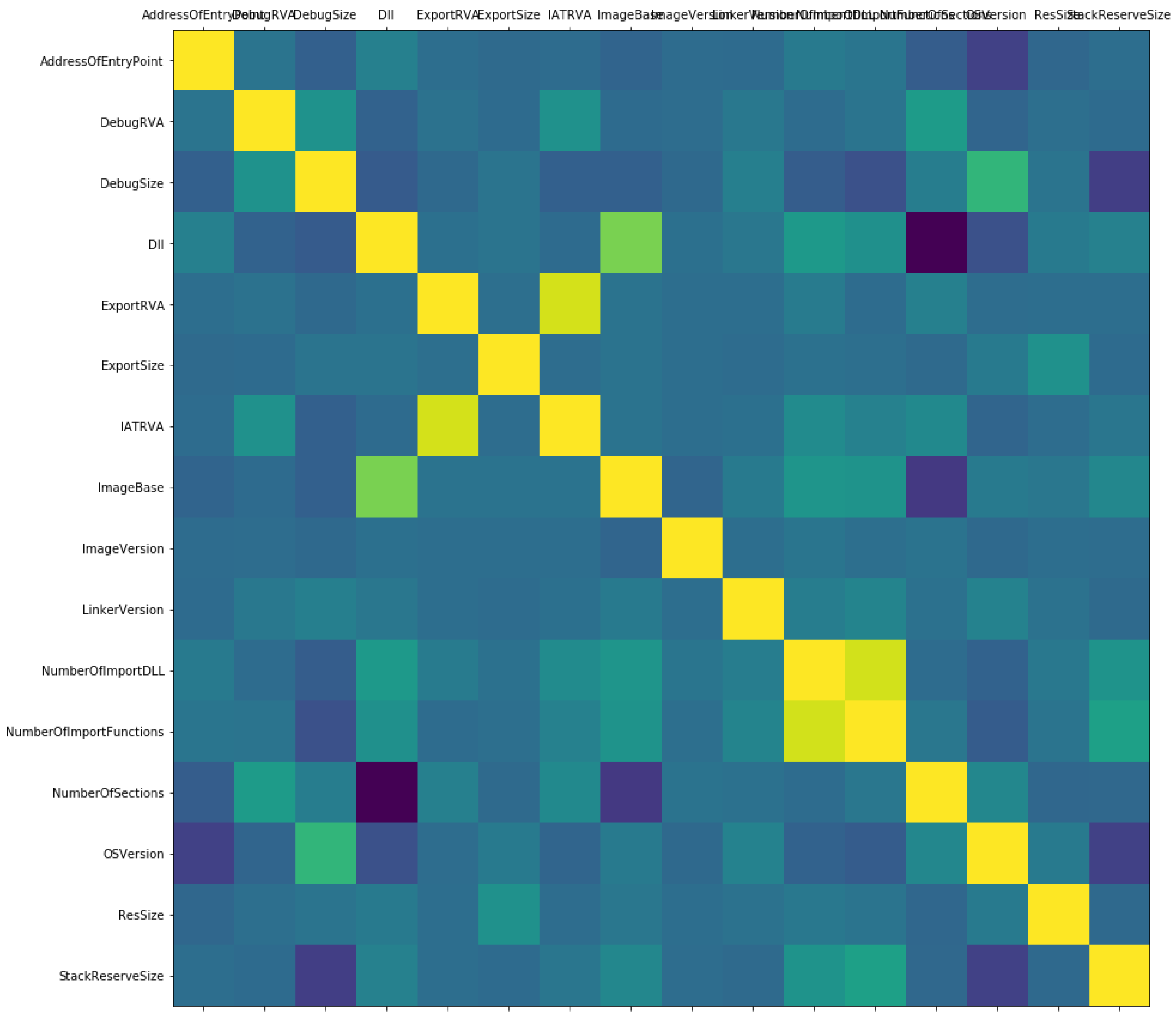


Histograms of Clean Dataframes

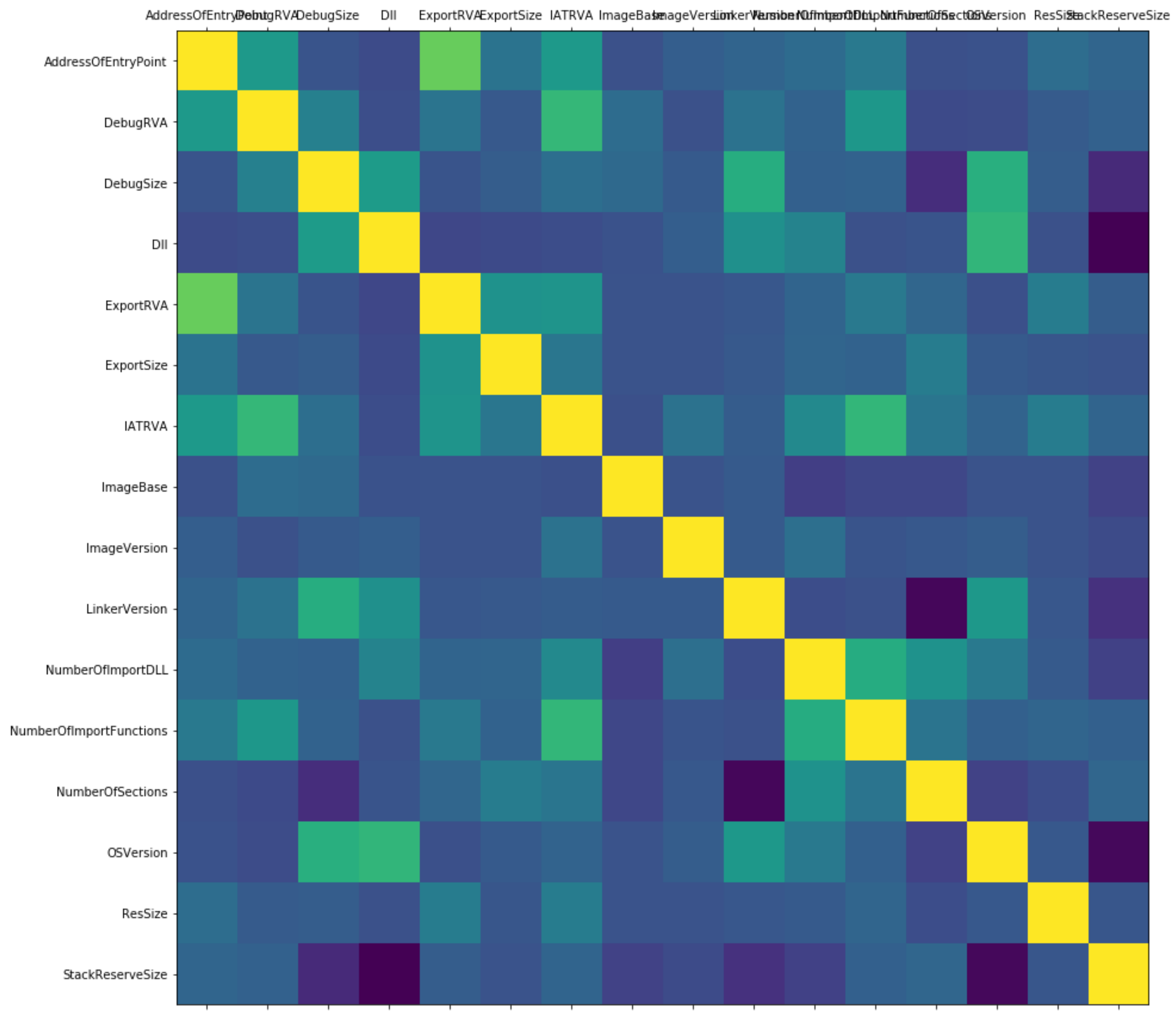




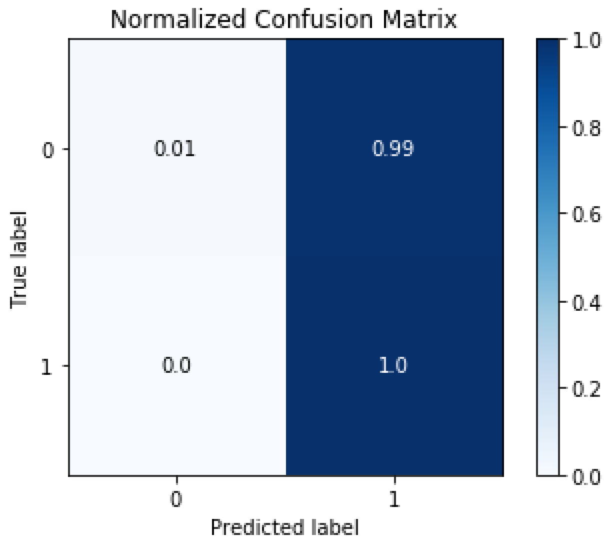
Malware Correlation


AddressOfEntryPoint
DebugRVA
DebugSize
Dll
ExportRVA
ExportSize
IATRVA
ImageBase
ImageVersion
LinkerVersion
NumberOfImportDLL
NumberOfImportFunctions
NumberOfSections
OSVersion
ResSize
StackReserveSize

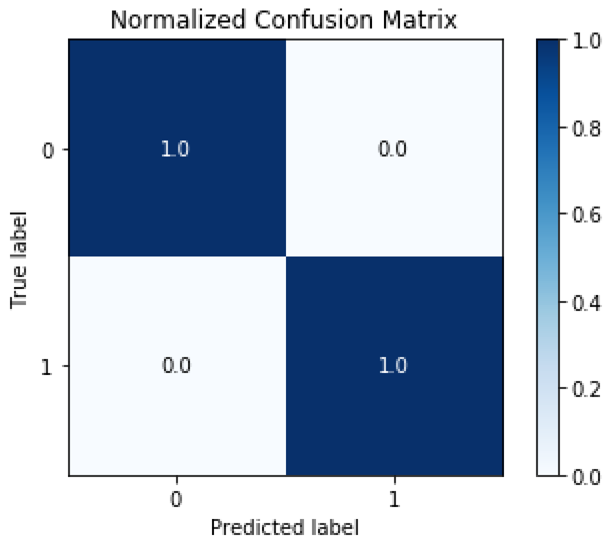
Clean Correlation



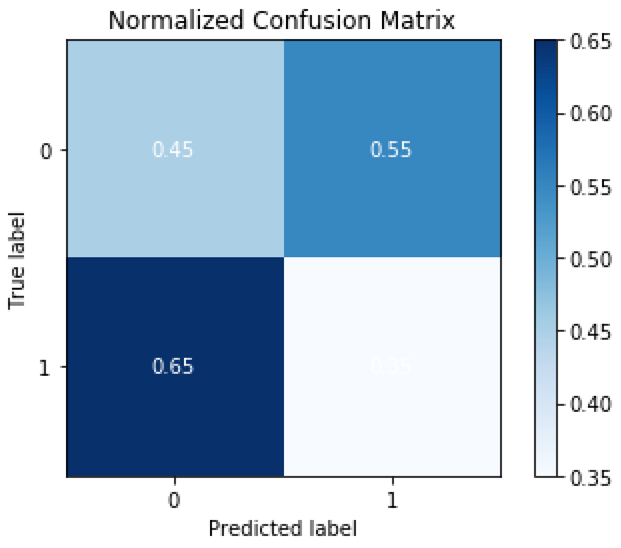
Naive Bayes Confusion Matrix



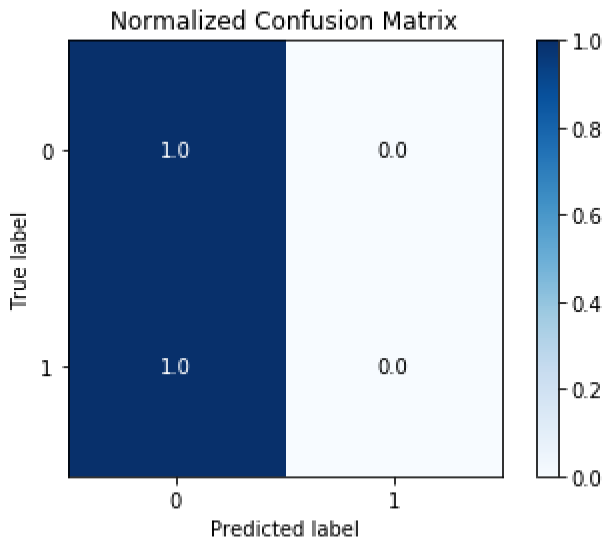
Random Forest Confusion Matrix



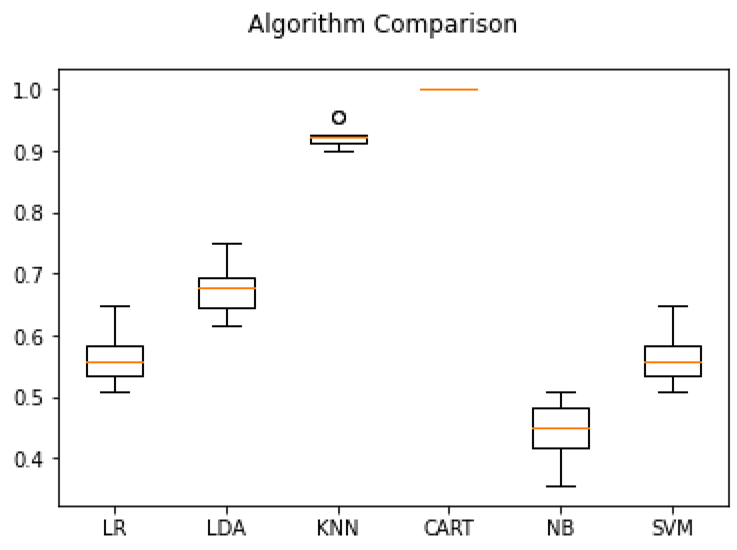
Logistic Regression Confusion Matrix



Logistic Regression CV Confusion Matrix



Algorithm Comparison



Conclusion

We get the following accuracies and Performance metrics for different models used:

Logistic Regression: 0.564841

```
Confusion Matrix
[[391   0]
 [297   0]]

Classification Report
              precision    recall  f1-score   support

           0       0.57      1.00      0.72       391
           1       0.00      0.00      0.00       297

    accuracy                           0.57       688
   macro avg       0.28      0.50      0.36       688
weighted avg       0.32      0.57      0.41       688
```

Linear Discriminant Analysis: 0.672675

Logistic Regression CV: 0.56

```
Confusion Matrix
[[391   0]
 [297   0]]

Classification Report
              precision    recall  f1-score   support

           0       0.57      1.00      0.72       391
           1       0.00      0.00      0.00       297

    accuracy                           0.57       688
   macro avg       0.28      0.50      0.36       688
weighted avg       0.32      0.57      0.41       688
```

K Nearest Neighbours: 0.923940

Naive Bayes: 0.446374

```
Confusion Matrix
[[  5 386]
 [  1 296]]

Classification Report
              precision    recall  f1-score   support

           0       0.83      0.01      0.03       391
           1       0.43      1.00      0.60       297

    accuracy                           0.44       688
   macro avg       0.63      0.50      0.31       688
weighted avg       0.66      0.44      0.28       688
```

Support Vector Machines: 0.564841

Random forest classifier: 1.00

```
Confusion Matrix
[[391   0]
 [  0 297]]

Classification Report
           precision    recall  f1-score   support

         0      1.00      1.00      1.00       391
         1      1.00      1.00      1.00       297

   accuracy                         1.00       688
  macro avg       1.00      1.00      1.00       688
weighted avg       1.00      1.00      1.00       688
```

So we observe that Random forest gives us the maximum accuracy followed by K-nearest neighbors whereas Naive Bayes Algorithm gives the lowest accuracy in detecting malware files among the clean and dirty ones.

Therefore, through this project, we were able to detect malware files among PE files using different machine learning algorithms and compare them on the basis of their accuracy.

REFERENCES

1. Anderson, H., Roth, P., “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models” arXiv:1804.04637v2 [cs.CR] 16 Apr 2018 - **https://arxiv.org/pdf/1804.04637.pdf**

2. Raff, E., Sylvester, J., Nicholas, C., “Learning the PE Header, Malware Detection with Minimal Domain Knowledge” arXiv.org pre-print of ACM AISec’17- **https://arxiv.org/ftp/arxiv/papers/1709/1709.01471.pdf**

3. Chau, D., Nachenberg, C. Wilhelm, J., Wright, A., Faloutsos, C., “Polonium: Tera-Scale Graph Mining for Malware Detection” **http://www.covert.io/research-papers/security/Polonium%20-%20Tera-Scale%20Graph%20Mining%20for%20Malware%20Detection.pdf**

4. Chaudhary P., Namita, “PE File-Based Malware Detection Using Machine Learning” **https://www.researchgate.net/publication/342640653_PE_File-Based_Malware_Detection_Using_Machine_Learning**

5. Raman, K., “Selecting Features to Classify Malware”- **https://www.covert.io/research-papers/security/Selecting%20Features%20to%20Classify%20Malware.pdf**

6. Comparative Study on Classic Machine learning Algorithms- **https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222**

7. Detect Malicious Executable(AntiVirus) Data Set- **https://archive.ics.uci.edu/ml/datasets/Detect+Malacious+Executable(AntiVirus)**

8. Free Malware Sample Sources for Researchers- **https://zeltser.com/malware-sample-sources/**

9. Awesome Machine Learning for Cyber Security- ***https://github.com/jivoi/awesome-ml-for-cybersecurity#-datasets***