

MALWARE DETECTION USING MACHINE LEARNING

A PROJECT REPORT

submitted by

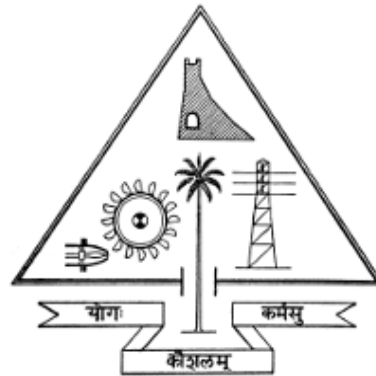
PRAFUL KUMAR (TCR18CS049)
JESTIN MATHEW (TCR18CS036)
DILDAS PM (TCR18CS026)

to

the APJ Abdul Kalam Technological University
in partial fulfilment of the requirements for the award of the Degree

of

Bachelor of Technology
in
Computer Science and Engineering



Department of Computer Science and Engineering

Government Engineering College, Thrissur
Thrissur – 680009

JUNE, 2022

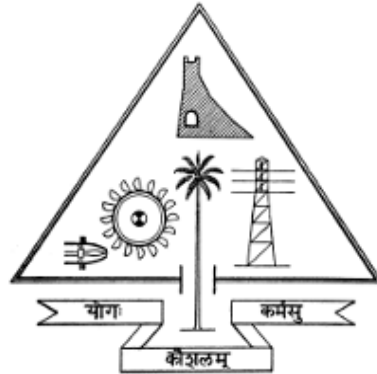
DECLARATION

I, on behalf of authors of the report: Praful Kumar, Jestin Mathew, Dildas P M, hereby declare that the project report “Malware Detection Using Machine Learning” submitted for partial fulfilment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University , Kerala is a bonafide work done by us under supervision of Mrs. jayasree. This submission represents our ideas in our own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also invoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place: Thrissur
Date: 01-06-2022

Author Project Leader

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GOVERNMENT ENGINEERING COLLEGE, THRISSUR
Thrissur – 680009



CERTIFICATE

This is to certify that the report entitled **‘Malware Detection Using Machine Learning’** submitted by **Praful (TCR18CS049), Dildas (TCR17CS026), Jestin (TCR17CS036)** to the APJ Abdul Kalam Technological University in partial fulfilment of the requirements for the award of the Degree of Bachelor Technology in Computer Science and Engineering is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Internal Supervisor(s)

Department Seal

Project Coordinator(s)

Head of the Department

ACKNOWLEDGEMENTS

It gives me great pleasure to present my project report on "Malware Diction Using Machine Learning". No work, however big or small, has ever been done without the contribution of others. So these words of acknowledgement come as a small gesture of gratitude towards all those people, without whom the successful completion of this report would not have been possible.

I am extremely grateful to Dr. Ranjini Bhattathiripad T, Principal, Government Engineering College Thrissur and Mrs Shibily Joseph, Associate Professor, Head of the Department, Computer Science and Engineering, for providing all the required resources for the successful completion of my seminar.

I would like to express my gratitude towards seminar coordinators Mr George Mathew, Assistant Professor, Department of Computer Science and Engineering who gave me their valuable suggestions, reviews, motivation and direction. I am greatly indebted to my guide, Mrs. Jayasree, Assistant Professor, Department of Computer Science and Engineering, for her unfailing guidance and immense support, without which I could not have made it this far. Last but not the least I would like to thank all my friends, who supported me with their valuable criticism, advice and support.

ABSTRACT

Malware Detection is a significant part of endpoint security including workstations, servers, cloud instances, and mobile devices. Malware Detection is used to detect and identify malicious activities caused by malware. With the increase in the variety of malware activities on different files online and offline, It's Important for Data Security, Privacy and protection. So, we will use Machine Learning and its algorithm to see the accuracy and prediction on Malware Datasets. In this Project we will use many different algorithms for analyzing and studying the Malware in Dataset.

We propose a versatile framework in which one can employ different machine learning algorithms to successfully distinguish between malware files and clean files, while aiming to minimise the number of false positives. In this paper we present the ideas behind our framework by working firstly with cascade one-sided perceptrons and secondly with cascade kernelized one-sided perceptrons. After having been successfully tested on medium-size datasets of malware and clean files, the ideas behind this framework were submitted to a scaling-up process that enable us to work with very large datasets of malware and clean files.

CONTENTS

Contents	Page No.
ACKNOWLEDGEMENTS	i
ABSTRACT	ii
LIST OF TABLES	iv
LIST OF FIGURES	v
CHAPTER 1. INTRODUCTION	1
1.1 Current System	1
1.2 Proposed System	3
1.3 Feasibility	5
1.3.1 Technical Feasibility	5
1.4 Process Model	5
1.5 MALWARE DETECTION CLASSIFICATION USING MA- CHINE LEARNING ALGORITHMS	6
CHAPTER 2. REQUIREMENT ANALYSIS	7
2.1 Method of requirement elicitation	7
2.2 User requirements	7
2.2.1 Software	7
2.3 Project requirements	8
2.4 The Portable Executable file format	8
CHAPTER 3. DESIGN AND IMPLEMENTATION	11
3.1 Architectural Description	11
3.2 Implementing the Model	11
3.2.1 Dataset	11
3.2.2 Algorithm	13
CHAPTER 4. CODING	19
4.1 Organization of Packages	19
CHAPTER 5. PERFORMANCE EVALUATION	22
5.1 Evaluation Scenario	22
5.1.1 Scenario based testing	22
5.2 Results	22
CHAPTER 6. DOCUMENTATION	23
6.1 Introduction	23
6.2 Installation	24
6.3 Working with the Product	24
6.4 Contact	24

CHAPTER 7. CONCLUSION AND FUTURE WORK	25
7.1 Conclusion	25
7.2 Advantages	26
7.3 Limitations & Future Expansions	26
REFERENCES	27

LIST OF TABLES

No.	Title	Page No.
-----	-------	----------

LIST OF FIGURES

No.	Title	Page No.
1.1	Machine learning workflow	2
1.2	Architecture	5
3.1	Number of files and Unique Combinations of Feature Values in the Training, Test, and Scale-Up Datasets.	12
3.2	Malware Distribution in the Training and Test Datasets.	12
3.3	Fitting Decision Tree Classifier to the training set	14
3.4	Decision Tree Parameter	15
3.5	Gradient Boosting final prediction formula	17
3.6	AdaBoost Model	17
4.1	Importing data-set	19
4.2	Algorithm testing and comparison	20
4.3	Best Algorithm	20
4.4	FALSE POSITIVE RATE AND FALSE NEGATIVE RATE	21
4.5	Save the algorithm and the feature list for later predictions	21
5.1	Result 1	22
5.2	Result 2	22

CHAPTER 1

INTRODUCTION

MALWARE is defined as software designed to infiltrate or damage a computer system without the owner's informed consent. Malware is actually a generic definition for all kinds of computer threats. A simple classification of malware consists of file infectors and stand-alone malware. Another way of classifying malware is based on their particular action: worms, backdoors, trojans, rootkits, spyware, adware etc. Malware detection through standard, signature based methods is getting more and more difficult since all current malware applications tend to have multiple polymorphic layers to avoid detection or to use side mechanisms to automatically update themselves to a newer version at short periods of time in order to avoid detection by any antivirus software. For an example of dynamical file analysis for malware detection, via emulation in a virtual environment, the interested reader can see [1]. Classical methods for the detection of metamorphic viruses are described in [2].

1.1 CURRENT SYSTEM

With growth of technology, the number of malware are also increasing. Malware now are designed with mutation characteristics which cause an enormous growth. Traditional signature based malware detection are proven to be ineffective against the vast variation of malware.

The success and consolidation of machine learning approaches would not have been possible without the confluence of three recent developments:

1. The first development is the increase in labeled feeds of malware meaning that, for the first time, labeled malware is available not only to the security community but also to the research community. The size of these feeds ranges from limited high-quality samples, like the ones provided by Microsoft (Ronen et al., 2018) for the Big Data Innovators Gathering Anti-Malware Prediction Challenge, to huge volumes of malware, such as theZoo (Yuval Nativ, 2015) or VirusShare (2011).
2. The second development is that computational power has increased rapidly and at the same time has become cheaper and closer to the budget of most researchers. Consequently, it allowed researchers to speed-up in the iterative training process and to fit larger and more complex models to the ever increasing data.
3. Third, the machine learning field has evolved at an increased pace during the last decades, achieving breakthrough success in terms of accuracy and scalability on a wide range of tasks, such as computer vision, speech recognition and natural language processing.

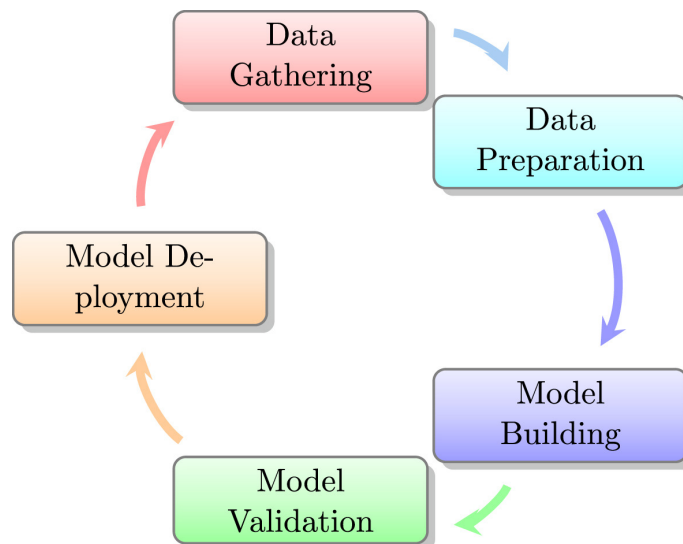


Figure 1.1: Machine learning workflow

In machine learning, a workflow is an iterative process that involves gathering available data, cleaning and preparing the data, building models, validating and deploying into production. See Fig. 2. Instead of dealing with raw malware, the data preparation process of traditional machine learning approaches involves pre-processing the executable to extract a set of features that provide an abstract view of the software. Afterwards the features are used to train a model to solve the task at hand. Because of the variety of malware functionalities, it is important not only to detect malicious software, but also to differentiate between different kinds of malware in order to provide a better understanding of their capabilities. The main difference between machine learning solutions for detection or classification of malware is the output returned by the system implemented. On the one hand, a malware detection system outputs a single value $y = f(x)$, in the range from 0 to 1, which indicates the maliciousness of the executable. On the other hand, a classification system outputs the probability of a given executable belonging to each output class or family, $\{p_1, p_2, \dots, p_N\}$, where N indicates the number of different families.

1.2 PROPOSED SYSTEM

Our system is basically divided into three major modules the first one is the user interface the second one is the train module and the third one is the malware test module.

The user interface module is the front-end module and this module basically contains the front-end architecture of the system. It basically provides an interface to the user for entering the file that is to be checked for malicious content. The next module is the train module. This module is used to train as well as test the selected models. The model to be

used is selected according to the accuracy of each.

This module is the main module and is responsible for the final classification result. In this module the classifier for the model Also gets generated.

The third module is the malware test module. This module is used to extract the data from the file that has been uploaded by the user through the user interface.

It is basically responsible for the extraction and determination of the data from the file, uploading and as well as the dividing of the data into various sections or features.

The architecture consists of mainly three modules that are: 1. Feature Database. 2. Feature Selection as well as Transformation and 3. Learning the Algorithms.

First, we discuss about the dataset, for this project we used Kaggle Microsoft malware classification challenge dataset which is a csv (comma separated file).

Then in next step there are some methods which are used to select the features like chi-square, information gain, fisher score, gain ratio and symmetric uncertainty feature selection methods.

After feature selection and transformation of dataset, it will split into two datasets: - first one is Testing Dataset and another is Training Dataset. In this we used various algorithms in detection of the unknown malware in the file.

Then the final step of the architecture is classification of the results, in proposed approach Random forest, Decision Tree, Linear Regression, Adaboost detect malware with much accuracy and improve the efficiency.

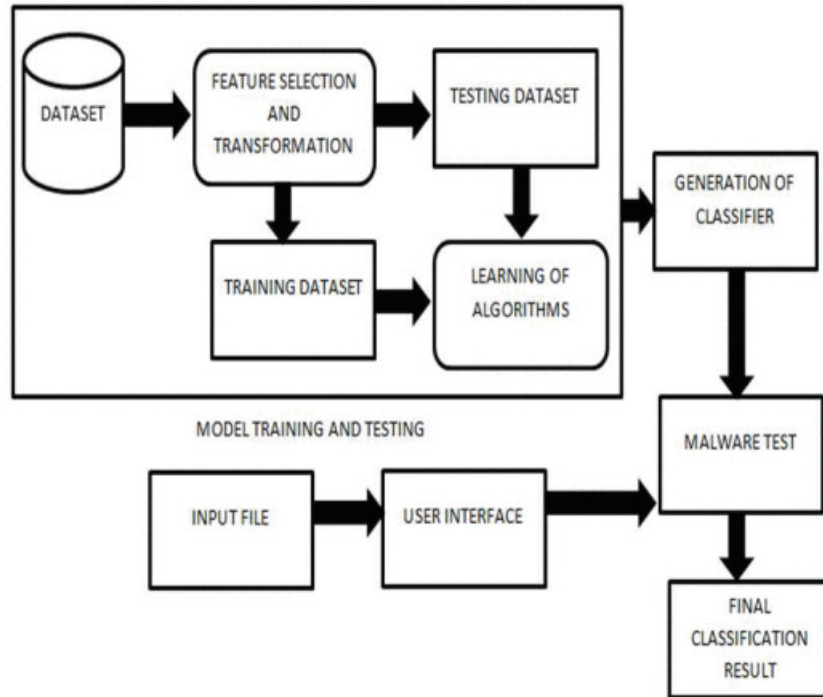


Figure 1.2: Architecture

1.3 FEASIBILITY

1.3.1 Technical Feasibility

Being very processor heavy, this can be implemented at enterprise gateway level to act as a central antivirus engine to supplement anti viruses present on end user computers.

1.4 PROCESS MODEL

Applying basic principles of Incremental Model, the project was planned to be designed, implemented and tested incrementally (a little more is added each time) until the product is finished. It involves both development of the device and maintenance. The product is defined as finished when it satisfies all of its requirements. The process model here aims to bring about a broader set of features until the targeted system is complete. Each

subsequent release of the project adds more complexity to the previous release until all the stages of project has been implemented.

1.5 MALWARE DETECTION CLASSIFICATION USING MACHINE LEARNING ALGORITHMS

Various Machine learning techniques are used for malware classification such as Support Vector Machine, Decision Tree, Naive Bayes, Random Forest, etc., and machine learning clustering techniques are used for clustering malware samples. The existing literature is discussed below about machine learning approaches for malware analysis proposed architecture for malware detection. In this architecture, first prepares the dataset which contains 140000 of files in which has 44000 clean files and 96000 million malware samples these are collected from different resources like VXheaven, Nothink, VirusShare, etc. After the collection of malware samples, automated malware analysis is performed on the collected dataset by using cuckoo sandbox which contains a series of python scripts to determine the behavior of malware during their execution.

CHAPTER 2

REQUIREMENT ANALYSIS

Requirements analysis and validation is a process of refinement, modeling and specification of the already discovered user requirements. The systematic use of proven principles, techniques, languages, and tools for analysis, documentation, and ongoing evolution of user needs and the specification of the external behavior of a system to satisfy those user needs form an integral part of this stage of development. This chapter describes the method of requirement elicitation employed and the user requirements thus gathered. The requirements are also finalized after validation using a suitable method.

2.1 METHOD OF REQUIREMENT ELICITATION

Understanding user requirements is an integral part of information systems design and is critical to the success of interactive systems. It is now widely understood that successful systems and products begin with an understanding of the needs and requirements of the users. As specified earlier we have decided to gather requirements from the related category type of end users. The different types of plans for requirements gathering are the following: ... Brainstorming Literature Survey Internet Surfing Observations

2.2 USER REQUIREMENTS

2.2.1 Software

Operating system: WIN, MAC OS, LINUX, etc IDE: Visual Studio Code, Jupyter notebook, Anaconda, pycharm, Google collab, etc Dataset: CSV Programming: Python

2.3 PROJECT REQUIREMENTS

One of the important part of our project is to minimize the false positive rate. So we will be training our algorithm with more than 1 lakhs of malware and testing different algorithm. We need to focus on which algorithm is quick and has least false positive rate. Since our project requires industrial computer to work more effectively and efficiently. **Data Preprocessing** The next step is conducting data preprocessing. The data preprocessing steps of this research are described as follows:

All the XML report files were parsed to select the most relevant and important attribute values (feature selection).

A term dictionary was created, which contains all the attribute values that were previously parsed and selected.

Each XML report file was compared against the term dictionary by counting the existence (or non-existence) of each term word in the term dictionary based on binary weight and term frequency weight.

Sparse vector models were created for each XML report file and Attribute-Relation File Format (ARFF) files were created.

2.4 THE PORTABLE EXECUTABLE FILE FORMAT

The Portable Executable (PE) format is a file format for executables, object code, DLLs, FON Font files and others used in 32-bit and 64-bit versions of the Windows operating system. The PE32 format stands for Portable Executables of 32-bit while PE32+ stands

for Portable Executables of 64-bit format.

Portable Executables encapsulate the information necessary for a Windows operating system to manage the executable code. This includes dynamic library references for linking, API export and import tables, resource management data and thread-local storage data. A PE file consists of a number of headers and sections that tell the dynamic linker how to map the file into memory. See Fig. 1. The PE Header contains information about the executable such as the number of sections, the size of the “PE Optional Header”, characteristics of the file, etc.² It also contains the import address table (IAT), which is a lookup table used by the application when calling a function in a different module. In addition, a Portable Executable file has various sections that contain the code and data of the executable including, but not limited to, the following:

- The .data section. This section is used to declare initialized data or constants that do not change at runtime.

1. The .bss section. This section is used for declaring variables and contains uninitialized data.
2. The .text section. This section keeps the actual code of the program.
3. The .rsrc section. This section contains all the resources of the program.
4. The .rdata section. This section holds the debug directory which stores the type, size and location of various types of debug information stored in the file.
5. The .idata section. This section contains information about functions and data that the program imports from DLLs.

6. The .edata section. This section contains the list of the functions and data that the PE file exports for other programs.
7. The .reloc section. This section holds a table of base relocations. A base relocation is an adjustment to an instruction or initialized variable value that is needed if the loader could not load the file where the linker assumed it would.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 ARCHITECTURAL DESCRIPTION

We will be importing sets of data set which is list if all kind of malware upload for different organisation. We will be preprocessing it and we will be training our algorithm with this data set.

3.2 IMPLEMENTING THE MODEL

3.2.1 Dataset

We used three datasets: a training dataset, a test dataset, and a “scale-up“ dataset. The number of malware files and respectively clean files in these datasets is shown in the first two columns of Table I. As stated above, our main goal is to achieve malware detection with only a few (if possible 0) false positives, therefore the clean files in this dataset (and also in the scale-up dataset) is much larger than the number of malware files.

From the whole feature set that we created for malware detection, 308 binary features were selected for the experiments to be presented in this paper. Files that generate similar values for the chosen feature set were counted only once. The last two columns in Table I show the total number of unique combinations of the 308 selected binary features in the training, test and respectively scale-up datasets. Note that the number of clean combinations – i.e combinations of feature values for the clean files – in the three datasets is much smaller than the number of malware unique combinations because most of the features were created to emphasize an aspect (either a geometrical form or behaviour aspect) of

malware file

	Files		Unique combinations	
Database	malware	clean	malware	clean
Training	27475	273133	7822	415
Test	11605	6522	506	130
Scale-up	approx. 3M	approx. 180M	12817	16437

Figure 3.1: Number of files and Unique Combinations of Feature Values in the Training, Test, and Scale-Up Datasets.

	Training Dataset		Test Dataset
Malware type	Files	Unique combinations of feature values	Files
Backdoor	35.52%	40.19%	9.16%
Hacktool	1.53%	1.73%	0.00%
Rootkit	0.09%	0.15%	0.04%
Trojan	48.06%	43.15%	37.17%
Worm	12.61%	12.11%	33.36%
Other malware	2.19%	2.66%	20.26%

Figure 3.2: Malware Distribution in the Training and Test Datasets.

The clean files in the training database are mainly system files (from different versions of operating systems) and executable and library files from different popular applications. We also use clean files that are packed or have the same form or the same geometrical similarities with malware files (e.g use the same packer) in order to better train and test the system.

The malware files in the training dataset have been taken from the Virus Heaven collection. The test dataset contains malware files from the WildList collection and clean files from different operating systems (other files than the ones used in the first database). The

malware collection in the training and test datasets consists of trojans, backdoors, hack-tools, rootkits, worms and other types of malware. The first and third columns in Table II represent the percentage of those malware types from the total number of files of the training and respectively test datasets. The second column in Table II represents the corresponding percentage of malware unique combinations from the total number of unique combinations of feature values for the training dataset. As shown in the first and last column of Table II, the distribution of malware types in this test data set is significantly different from the malware distribution in the training data-set.

3.2.2 Algorithm

(i) Decision Tree

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

1. Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
2. The logic behind the decision tree can be easily understood because it shows a tree-like structure.

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree.

The complete process can be better understood using the below algorithm:

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the Decision Tree Classifier class from sklearn.tree library. Below is the code for it:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

Figure 3.3: Fitting Decision Tree Classifier to the training set

In the above code, we have created a classifier object, in which we have passed two main parameters;

- `criterion='entropy'`: Criterion is used to measure the quality of split, which is calculated by information gain given by entropy
- `random state=0`: For generating the random states.

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
                        random_state=0, splitter='best')
```

Figure 3.4: Decision Tree Parameter

(ii) Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as Decision tree Classifier, KNN, SVM, Logistic Regression, etc. Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)

(iii) Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees). The ensemble consists of N trees. Tree1 is trained using the feature matrix X and the labels y . The predictions labelled $y_1(\text{hat})$ are used to determine the training set residual errors r_1 . Tree2 is then trained using the feature matrix X and the residual errors r_1 of Tree1 as labels. The predicted results $r_1(\text{hat})$ are then used to determine the residual r_2 . The process is repeated until all the N trees forming the ensemble are trained.

There is an important parameter used in this technique known as Shrinkage.

Shrinkage refers to the fact that the prediction of each tree in the ensemble is shrunk after it is multiplied by the learning rate (η) which ranges between 0 to 1. There is a trade-off between η and number of estimators, decreasing learning rate needs to be

compensated with increasing estimators in order to reach certain model performance. Since all trees are trained now, predictions can be made. Each tree predicts a label and final prediction is given by the formula, The class of the gradient boosting regression in

$$y(\text{pred}) = y_1 + (\text{eta} * r_1) + (\text{eta} * r_2) + \dots + (\text{eta} * r_N)$$

Figure 3.5: Gradient Boosting final prediction formula

scikit-learn is GradientBoostingRegressor. A similar algorithm is used for classification known as GradientBoostingClassifier.

(iv) AdaBoost

AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique that combines multiple “weak classifiers” into a single “strong classifier”. Let’s create the AdaBoost Model using Scikit-learn. AdaBoost uses Decision Tree Classifier as default Classifier.

```
# Create adaboost classifier object
abc = AdaBoostClassifier(n_estimators=50,
                        learning_rate=1)
# Train Adaboost Classifier
model = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = model.predict(X_test)
```

Figure 3.6: AdaBoost Model

The most important parameters are base estimator, n estimators, and learning rate."

- base estimator: It is a weak learner used to train the model. It uses Decision-TreeClassifier as default weak learner for training purpose. You can also specify different machine learning algorithms.
- n_estimators: Number of weak learners to train iteratively.
- learning_rate: It contributes to the weights of weak learners. It uses 1 as a default value.

CHAPTER 4

CODING

4.1 ORGANIZATION OF PACKAGES

Since we will be using python and machine learning to implement our code so we will need to import all the libraries like sklearn, joblib and many more. Then we will be importing our data set to check our algorithm is working or not. Once we train a model we will cross validate the best model and check PE file whether it is malware or not. Machine learning algorithms were trained on datasets to discern if requested permissions or intent filters are more effective indicators of malware. One was of only intent filters, one was only requested and declared permissions, and one was both feature sets combined.

- IMPORTING DATA-SET

```
data = pd.read_csv('data.csv', sep='|')
X = data.drop(['Name', 'md5', 'legitimate'], axis=1).values
y = data['legitimate'].values
X.shape, y.shape
```

Figure 4.1: Importing data-set

- ALGORITHM TESTING AND COMPARISON

```
#Algorithm comparison
algorithms = {
    "DecisionTree": tree.DecisionTreeClassifier(max_depth=10),
    "RandomForest": ske.RandomForestClassifier(n_estimators=50),
    "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=50),
    "AdaBoost": ske.AdaBoostClassifier(n_estimators=100),
    "GNB": GaussianNB()
}

results = {}
print("\nNow testing algorithms")
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("%s : %f %" % (algo, score*100))
    results[algo] = score
```

Figure 4.2: Algorithm testing and comparison

- BEST ALGORITHM

```
winner = max(results, key=results.get)
print('\nWinner algorithm is %s with a %f %% success' % (winner, results[winner]*100))

Winner algorithm is RandomForest with a 99.460340 % success
```

Figure 4.3: Best Algorithm

- FALSE POSITIVE RATE AND FALSE NEGATIVE RATE

```
print("False positive rate : %f %" % ((mt[0][1] / float(sum(mt[0])))*100))
print('False negative rate : %f %' % ( (mt[1][0] / float(sum(mt[1]))*100))

False positive rate : 0.396662 %
False negative rate : 0.878263 %
```

Figure 4.4: FALSE POSITIVE RATE AND FALSE NEGATIVE RATE

- Save the algorithm and the feature list for later predictions

```
# Save the algorithm and the feature list for later predictions
print('Saving algorithm and feature list in classifier directory...')
joblib.dump(algorithms[winner], 'classifier/classifier.pkl')
joblib.dump(features, 'classifier/features.pkl')
# open('classifier/features.pkl', 'w').write(pickle.dumps(features))
print('Saved')

Saving algorithm and feature list in classifier directory...
Saved
```

Figure 4.5: Save the algorithm and the feature list for later predictions

Now we have trained are algorithm as shown above now we will use this algorithm and we will try to fin out whether the application installed on our PC is safe or not.

CHAPTER 5

PERFORMANCE EVALUATION

5.1 EVALUATION SCENARIO

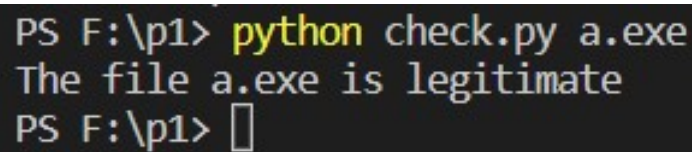
5.1.1 Scenario based testing

As we loaded our data set which has more than 1 lakh malware we initially separated our malware. Then we grouped our data which is malware or not.

5.2 RESULTS

The output of our project was is show below.

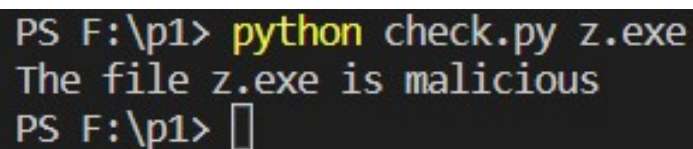
- A.EXE IS FILE WHICH WE WILL RUN AND CHECK WHETHER IT IS MALWARE OR NOT.



```
PS F:\p1> python check.py a.exe
The file a.exe is legitimate
PS F:\p1> 
```

Figure 5.1: Result 1

- Z.EXE IS FILE WHICH WE WILL RUN AND CHECK WHETHER IT IS MALWARE OR NOT.



```
PS F:\p1> python check.py z.exe
The file z.exe is malicious
PS F:\p1> 
```

Figure 5.2: Result 2

CHAPTER 6

DOCUMENTATION

6.1 INTRODUCTION

In order to keep our systems safe we need to ensure that there are no files which contain any kind of malware. So we implemented our system in order to detect such kind of malware. While implementing the train module we applied multiple algorithms on a dataset in order to achieve the best possible accuracy for our system. We selected the model with the best accuracy for a dataset.

So according to the results achieved that was shown in the table it can be determined that the random forest provides the best accuracy on the dataset. Hence the classifier for random forest is generated.

This shows that the accuracy achieved for our system is about 99 percent which is good accuracy in order to detect the malware. So it can be described that the results that were produced by your system are the accuracy which is 99 percent, false positive rate which is 0.104 percent, false negative rate which is 0.154 percent.

A classification approach can be additionally implemented for the malware detection system presented which will involve the correct identification of the type of malware that has attacked the file and can be used as a base for different researches in order to identify the most commonly attacking malwares. So, this presents an idea about the future work for the project that can be implemented. Since most AntiVirus products manage to have a detection rate of over 90 percent, it follows that an increase of the total detection rate of 3 percent – 4 percent as the one produced by our algorithms, is very significant. (Note

that the training is performed on the malware samples that are not detected by standard detection methods.)

6.2 INSTALLATION

6.3 WORKING WITH THE PRODUCT

Our product consist of machine learning algorithm which check the data set and by using different algorithm for maximum efficiency.

6.4 CONTACT

For any enquiries ,regarding any aspects of this product, can be done using the following contact.

- **Praful Kumar:** email- prafulkumar100@gmail.com
- **Jestin mathew:** email- jestinmathew1221@gmail.com
- **Dildas PM:** email- dildas221thrissur@gmail.com

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Our main target was to come up with a machine learning framework that generically detects as much malware samples as it can, with the tough constraint of having a zero false-positive rate. We were very close to our goal, although we still have a non-zero false positive rate. In order that this framework to become part of a highly competitive commercial product, a number of deterministic exception mechanisms have to be added. In our opinion, malware detection via machine learning will not replace the standard detection methods used by anti-virus vendors, but will come as an addition to them.

Advantage of making such an effective software is that, in this world every day new malware is getting invented. We need such an algorithm that can work against all kind of malware whether it is new or old. Advantage of our algorithm is that we have trained with more than 1 lakh malware compared to other they usually train their algorithm with maximum 1000 data set.

A classification approach can be additionally implemented for the malware detection system presented which will involve the correct identification of the type of malware that has attacked the file and can be used as a base for different researches in order to identify the most commonly attacking malwares. So, this presents an idea about the future work for the project that can be implemented.

7.2 ADVANTAGES

Advantage of making such a effective software is that ,in this world every day new malware is getting invented. We need such an algorithm that can work against all kind of malware whether it is new or old. Advantage of our algorithm is that we have trained with with more than 1 lakh malware compared to other they usually train there algorithm with maximum 1000 data set.

7.3 LIMITATIONS & FUTURE EXPANSIONS

Major limitation of our software is that we need industrial computer to train our algorithm more effectively.

The classifier only depends on the manifest file. Even if the classifier is effective now, the malware landscape is constantly shifting and may render the approach obsolete in a few years. In general static approaches can be fooled by code obfuscation.

REFERENCES

- [1]<http://www.us-cert.gov/controlsystems/pdf/undirectedattack0905.pdf>
- [2] "Defining Malware: FAQ". <http://technet.microsoft.com>. Retrieved 2009-09-10.
- [3] F-Secure Corporation (December 4, 2007). "F-Secure Reports Amount of Malware Grew by 100". Retrieved 2007-12-11.
- [4] Filiol, E., Jacob, G., Liard, M.L., 2007: Evaluation methodology and theoretical model for antiviral behavioral detection strategies. J. Comput. 3, pp 27–37.