Malware Detection System Using Machine Learning

MGT3002 – Information and System Security

Winter Semester 2021-2022

Slot – E2

FINAL REPORT

Submitted by

Dhivyadharshini B R	- 19MIS0185
M. Sreeram	- 19MIS0283

Under the guidance of

Prof. Mangayakarasi R

Abstract

Malware is today one of the biggest security threats to the Internet. Malware is any malicious software with the intent to perform malevolent activities on a targeted system. Viruses, worms, trojans, backdoors and adware are but a few examples that fall under the umbrella of malware. Over the last decade, a war has been taking place which involves the computer security community and the black hat hackers. The security community uses every possible routine and strategies in order to cease and also to remove the damage caused by the malicious programs while at the same time the black hat community advances their techniques that can easily bypass the installed security measures.

In this Report we present a Machine Learning approach for classifying a file as Malicious or Legitimate, testing and comparison of various machine learning methods for Malware detection.

Problem Statement

With the growth of technology, the number of malwares is also increasing day by day. Malware now are designed with mutation characteristic which causes an enormous growth in number of the variation of malware. Not only that, with the help of automated malware generated tools, novice malware author is now able to easily generate a new variation of malware. With these growths in new malware, traditional signature-based malware detection is proven to be ineffective against the vast variation of. On the other hand, machine learning methods for malware detection are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate for detecting malware.

The extent of the damage caused by malicious software will often depend on whether the malware has infected a home computer or a corporate network. Whereas, in some cases the results of a malware infection may be imperceptible to the user, in other cases the damage can have serious consequences:

- For home users, an infection may involve the loss of relatively unimportant information that can be easily replaced, or it could result in the loss of information that gives the cybercriminal access to the user's bank account.
- On a corporate network, a Trojan virus that sends spam may generate a minor increase in communications traffic, whereas other types of infection could result in the complete breakdown of the corporate network or the loss of business-critical data.

Malware Detection System

Among the more familiar forms of automated monitoring technologies, malware detection comprises mechanisms to identify and protect against harm from viruses, worms, Trojan horses, spyware, and other forms of malicious code. Malware detection and prevention technologies are widely available for servers, gateways, user workstations, and mobile devices, with some tools offering the capability to centrally monitor malware detection software installed on multiple systems or computers. Malware detection tools typically run continuously and provide automated updates of detection signatures or other reference information used to identify malicious code.

- **Malware detection** refers to the process of detecting the presence of malware on a host system or of distinguishing whether a specific program is malicious or legitimate.
- **Malware** is a threat to the computer users regardless which operating systems and hardware platforms that they are using.
- It is an **intrusive software** that is designed to cause damage to a computer, server, client, or computer network. Malware is a contraction for "malicious

Malware Detection Techniques

- **Signatured Based:** The signature-based detection technique is used by most of the antivirus programs. The antivirus program disassembles the code of the infected file and search for the pattern that belong to a malware family. Signatures of the malwares are maintained in database and then further used for comparison in detection process. This kind of detection technique is also known as string or pattern scanning or matching. It can be static, dynamic or hybrid as well.
- **Heuristic Based:** The heuristic-based detection detects or differentiate between the normal and abnormal behaviour of a system
  so that ultimately the known and unknown malware attacks can be identified and resolved.
  Heuristic based technique consist of the following three basic components.
    - Data collection
    - Interpretation
    - Matching algorithm

In Our Project we have used Heuristic Based Malware Detection

Modules

- **Data Collection**
  We downloaded the dataset from KAGGLE, and we have used that data for malware analysis and classification.
- **Feature Identification**
  Feature Selection and Identification is one of the core concepts in machine learning which hugely impacts the performance of your model. The data features that we use to train our machine learning models have a huge influence on the performance you can achieve.
  Irrelevant or partially relevant features can negatively impact model performance. Feature selection and Data cleaning should be the first and most important step of your model designing. In our system we have used ExtraTreesClassifier for feature identification
- **Building Machine Learning model**
  We built a machine learning model in which the approach tries out 6 different classification algorithms before deciding which one to use for prediction by

comparing their results. Different Machine Learning models tried are, Linear Regression, Random Forest, Decision Tree, Adaboost, Gaussian, Gradient Boosting.

- **Training Model**
  In this we train each machine model that we have used in our system using with the X_train and testing with X_test.
  Finally, the model with best accuracy will be ranked as winner
- **File Testing**
  To test the model on an unseen file, it's required to extract the characteristics of the given file. Python's pefile.PE library is used to construct and build the feature vector and a ML model is used to predict the class for the given file based on the already trained model. We developed a python code in which it extracts all the characteristics of the given file and classifies whether the given input file is malicious or legitimate.

Proposed Technique

This approach tries out 6 different classification algorithms before deciding which one to use for prediction by comparing their results. Different Machine Learning models tried are, Linear Regression, Random Forest, Decision Tree, Adaboost, Gaussian, Gradient Boosting.

In order to test the model on an unseen file, it's required to extract the characteristics of the given file. Python's pefile.PE library is used to construct and build the feature vector and a ML model is used to predict the class for the given file based on the already trained model.

Step1: Importing all the required libraries

Step2: Loading the initial dataset delimited by |

Step3: Extracting Number of malicious files vs Legitimate files in the training set

Step4: Dropping columns like Name of the file, MD5 (message digest) and label

Step5: Classifying using ExtraTreesClassifier – [ExtraTreesClassifier fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting]

Step6: Display Number of Features

Step7: Cross Validation (Cross validation is applied to divide the dataset into random train and test subsets. test_size = 0.2 represent the proportion of the dataset to include in the test split)

Step8: Display Features identified by ExtraTreesClassifier

Step9: Building Machine Learning Model

Step10: Training each of the model with the X_train and testing with X_test. The model with best accuracy will be ranked as winner

Step11: Saving the Model

Step12: Calculating False positive and False negative on the data set

Step13: Testing Files with the best accuracy model and checking whether it is malicious or legitimate

Methodologies (Machine Learning Models)

Linear Regression:

Linear Regression is one of the simplest Machine learning algorithms that comes under Supervised Learning technique and used for solving regression problems.

It is used for predicting the continuous dependent variable with the help of independent variables.

The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable.

If single independent variable is used for prediction, then it is called Simple Linear Regression and if there are more than two independent variables then such regression is called as Multiple Linear Regression.

Random Forest

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.

It is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. Decisions or the test are performed on the basis of features of the given dataset.

Adaboost

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. AdaBoost was the first 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”.

Gaussian

Gaussian processes are a powerful algorithm for both regression and classification. Their greatest practical advantage is that they can give a reliable estimate of their own uncertainty. Gaussian processes let us describe probability distributions over functions.

Gradient Boosting.

Gradient boosting is a method standing out for its prediction speed and accuracy, particularly with large and complex datasets. The main idea behind this algorithm is to build models sequentially and these subsequent models try to reduce the errors of the previous model. This is done by building a new model on the errors or residuals of the previous model.

Extra Trees Classifier

Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest.

Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

Implementation

Dataset: https://www.kaggle.com/datasets/nsaravana/malware-detection/101639

Attributes: 'Name', 'md5', 'Machine', 'SizeOfOptionalHeader', 'Characteristics',

'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',

'SizeOfInitializedData', 'SizeOfUninitializedData',

'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase',

'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion',

'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',

'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage',

'SizeOfHeaders', 'CheckSum', 'Subsystem', 'DllCharacteristics',

'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',

'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'SectionsNb', 'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',

'SectionsMeanRawsize', 'SectionsMinRawsize', 'SectionMaxRawsize',

'SectionsMeanVirtualsize', 'SectionsMinVirtualsize',

'SectionMaxVirtualsize', 'ImportsNbDLL', 'ImportsNb',

'ImportsNbOrdinal', 'ExportNb', 'ResourcesNb', 'ResourcesMeanEntropy',

'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'ResourcesMeanSize',

'ResourcesMinSize', 'ResourcesMaxSize', 'LoadConfigurationSize',

'VersionInformationSize', 'legitimate'

Here We have use two different systems. One for analysing only the dataset and another for analysing the dataset as well as classifying whether the given input file is malicious or legitimate

MODEL1

Analysing Dataset (Malware-classification.ipynb)

```
import os

import pandas

import numpy

import pickle

import sklearn.ensemble as ek

from sklearn.model_selection import train_test_split

from sklearn import tree, linear_model

from sklearn.feature_selection import SelectFromModel

import joblib

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import confusion_matrix

from sklearn.pipeline import make_pipeline

from sklearn import preprocessing

from sklearn import svm

from sklearn.linear_model import LinearRegression

dataset = pandas.read_csv('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/data.csv',sep='|', low_memory=False)

dataset.head()
```

```
dataset.describe()
dataset.groupby(dataset['legitimate']).size()
X = dataset.drop(['Name','md5','legitimate'],axis=1).values
y = dataset['legitimate'].values
extratrees = ek.ExtraTreesClassifier().fit(X,y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
nbfeatures
X_train, X_test, y_train, y_test = train_test_split(X_new, y ,test_size=0.2)
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]],
extratrees.feature_importances_[index[f]]))
    features.append(dataset.columns[2+f])

model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
        "RandomForest":ek.RandomForestClassifier(n_estimators=50),
        "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
        "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
        "GNB":GaussianNB(),
        "LinearRegression":LinearRegression()
}
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score
```

```
winner = max(results, key=results.get)

joblib.dump(model[winner],'E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/classifier.pkl')

open('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/features.pkl', 'wb').write(pickle.dumps(features))

clf = model[winner]

res = clf.predict(X_new)

mt = confusion_matrix(y, res)

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)))

# Load classifier

clf = joblib.load('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/classifier.pkl')

#load features

features = pickle.loads(open(os.path.join('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/features.pkl'),'rb').read())
```

MODEL2:

main.py

```
import os

import time

import pyfiglet


def run_PE():

    file = input("Enter the path and name of the file : ")

    os.system("python Extract/PE_main.py {}".format(file))


def run_URL():

    os.system('python Extract/url_main.py')


def exit():

    os.system('exit')
```

```
def start():
    print(pyfiglet.figlet_format("Malware Detector"))
    print(" Welcome to antimalware detector \n")
    print(" 1. PE scanner")
    print(" 2. Exit\n")

    select = int(input("Enter your choice : "))

    if (select in [1,2]):

        if(select == 1):
            run_PE()
            choice = input("Do you want to search again? (y/n)")
            if(choice not in ['Y','N','n','y']):
                print("Bad input\nExiting...")
                time.sleep(3)
                exit()
            else:
                if(choice == 'Y' or 'y'):
                    start()
                elif(choice == 'N' or 'n'):
                    exit()

        else:
            exit()
    else:
        print("Bad input\nExiting...")
        time.sleep(3)
        exit()
```

start()

ML_Classification (analysing dataset – PE.ipynb)

import pandas as pd

dataset = pd.read_csv(r"C:\Users\sreeram\Desktop\Malware-Detection-using-Machine-learning-main\Dataset\data.csv", sep='|')

dataset.head()   #Top 5 row of the dataset

dataset.tail()   #Last 5 row of the dataset

dataset.columns   # name of the columns

dataset.describe(include="all")   # summary of numeric attributes

dataset.info()   # info about the whole dataset

dataset["legitimate"].value_counts()   # count of malware (0) and benign (1) files in dataset

import matplotlib.pyplot as plt

dataset["legitimate"].value_counts().plot(kind="pie",autopct="%1.1f%%")

plt.show()

import os

import pandas

import numpy

import pickle

import pefile

import sklearn.ensemble as ek

from sklearn.feature_selection import SelectFromModel

import joblib

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.linear_model import LinearRegression

from sklearn.metrics import confusion_matrix

from sklearn import svm

import sklearn.metrics as metrics

Feature

```
X = dataset.drop(['Name','md5','legitimate'],axis=1).values    #Droping this because
classification model will not accept object type elements (float and int only)

# Target variable

y = dataset['legitimate'].values

extratrees = ek.ExtraTreesClassifier().fit(X,y)

model = SelectFromModel(extratrees, prefit=True)

X_new = model.transform(X)

nbfeatures = X_new.shape[1]

#Number of important features

nbfeatures

#splitting the data (70% - training and 30% - testing)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_new, y ,test_size=0.29, stratify = y)

features = []

index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]

for f in range(nbfeatures):

   print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]],
extratrees.feature_importances_[index[f]]))

   features.append(dataset.columns[2+f])

model = { "DecisionTree": DecisionTreeClassifier(max_depth=10),

       "RandomForest":ek.RandomForestClassifier(n_estimators=50),

       "Adaboost":ek.AdaBoostClassifier(n_estimators=50),

       "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),

       "GNB":GaussianNB(),

       "LinearRegression":LinearRegression(),

       }

results = {}

for algo in model:

   clf = model[algo]

   clf.fit(X_train,y_train)

   score = clf.score(X_test,y_test)
```

```
    print ("%s : %s " %(algo, score))
    results[algo] = score
winner = max(results, key=results.get)# Selecting the classifier with good result
print("Using", winner, "for classification, with",len(features), 'features.')
joblib.dump(model[winner],'classifier.pkl')
open('features.pkl', 'wb').write(pickle.dumps(features))
# Load classifier
clf = joblib.load('classifier.pkl')
#load features
features = pickle.loads(open(os.path.join('features.pkl'),'rb').read())
```

PE_main.py

In this program we are first extracting the features from the PE and then providing it to the saved machine and using thoses features we are prediciting whether the PE is malicious or not.

```
'''


import pefile
import os
import array
import math
import pickle
import joblib
import sys
import argparse



#For calculating the entropy
def get_entropy(data):
    if len(data) == 0:
        return 0.0
```

```
    occurences = array.array('L', [0]*256)
    for x in data:
        occurences[x if isinstance(x, int) else ord(x)] += 1

    entropy = 0
    for x in occurences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x*math.log(p_x, 2)

    return entropy

#For extracting the resources part
def get_resources(pe):
    """Extract resources :
    [entropy, size]"""
    resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in resource_id.directory.entries:
                                data = pe.get_data(resource_lang.data.struct.OffsetToData,
resource_lang.data.struct.Size)
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)

                                resources.append([entropy, size])
        except Exception as e:
```

```
            return resources
    return resources

#For getting the version information
def get_version_info(pe):
    """Return version infos"""
    res = {}
    for fileinfo in pe.FileInfo:
        if fileinfo.Key == 'StringFileInfo':
            for st in fileinfo.StringTable:
                for entry in st.entries.items():
                    res[entry[0]] = entry[1]
        if fileinfo.Key == 'VarFileInfo':
            for var in fileinfo.Var:
                res[var.entry.items()[0][0]] = var.entry.items()[0][1]
    if hasattr(pe, 'VS_FIXEDFILEINFO'):
          res['flags'] = pe.VS_FIXEDFILEINFO.FileFlags
          res['os'] = pe.VS_FIXEDFILEINFO.FileOS
          res['type'] = pe.VS_FIXEDFILEINFO.FileType
          res['file_version'] = pe.VS_FIXEDFILEINFO.FileVersionLS
          res['product_version'] = pe.VS_FIXEDFILEINFO.ProductVersionLS
          res['signature'] = pe.VS_FIXEDFILEINFO.Signature
          res['struct_version'] = pe.VS_FIXEDFILEINFO.StrucVersion
    return res

#extract the info for a given file using pefile
def extract_infos(fpath):
    res = {}
    pe = pefile.PE(fpath)
    res['Machine'] = pe.FILE_HEADER.Machine
    res['SizeOfOptionalHeader'] = pe.FILE_HEADER.SizeOfOptionalHeader
```

```
    res['Characteristics'] = pe.FILE_HEADER.Characteristics
    res['MajorLinkerVersion'] = pe.OPTIONAL_HEADER.MajorLinkerVersion
    res['MinorLinkerVersion'] = pe.OPTIONAL_HEADER.MinorLinkerVersion
    res['SizeOfCode'] = pe.OPTIONAL_HEADER.SizeOfCode
    res['SizeOfInitializedData'] = pe.OPTIONAL_HEADER.SizeOfInitializedData
    res['SizeOfUninitializedData'] = pe.OPTIONAL_HEADER.SizeOfUninitializedData
    res['AddressOfEntryPoint'] = pe.OPTIONAL_HEADER.AddressOfEntryPoint
    res['BaseOfCode'] = pe.OPTIONAL_HEADER.BaseOfCode
    try:
        res['BaseOfData'] = pe.OPTIONAL_HEADER.BaseOfData
    except AttributeError:
        res['BaseOfData'] = 0
    res['ImageBase'] = pe.OPTIONAL_HEADER.ImageBase
    res['SectionAlignment'] = pe.OPTIONAL_HEADER.SectionAlignment
    res['FileAlignment'] = pe.OPTIONAL_HEADER.FileAlignment
    res['MajorOperatingSystemVersion'] =
pe.OPTIONAL_HEADER.MajorOperatingSystemVersion
    res['MinorOperatingSystemVersion'] =
pe.OPTIONAL_HEADER.MinorOperatingSystemVersion
    res['MajorImageVersion'] = pe.OPTIONAL_HEADER.MajorImageVersion
    res['MinorImageVersion'] = pe.OPTIONAL_HEADER.MinorImageVersion
    res['MajorSubsystemVersion'] = pe.OPTIONAL_HEADER.MajorSubsystemVersion
    res['MinorSubsystemVersion'] = pe.OPTIONAL_HEADER.MinorSubsystemVersion
    res['SizeOfImage'] = pe.OPTIONAL_HEADER.SizeOfImage
    res['SizeOfHeaders'] = pe.OPTIONAL_HEADER.SizeOfHeaders
    res['CheckSum'] = pe.OPTIONAL_HEADER.CheckSum
    res['Subsystem'] = pe.OPTIONAL_HEADER.Subsystem
    res['DllCharacteristics'] = pe.OPTIONAL_HEADER.DllCharacteristics
    res['SizeOfStackReserve'] = pe.OPTIONAL_HEADER.SizeOfStackReserve
    res['SizeOfStackCommit'] = pe.OPTIONAL_HEADER.SizeOfStackCommit
    res['SizeOfHeapReserve'] = pe.OPTIONAL_HEADER.SizeOfHeapReserve
```

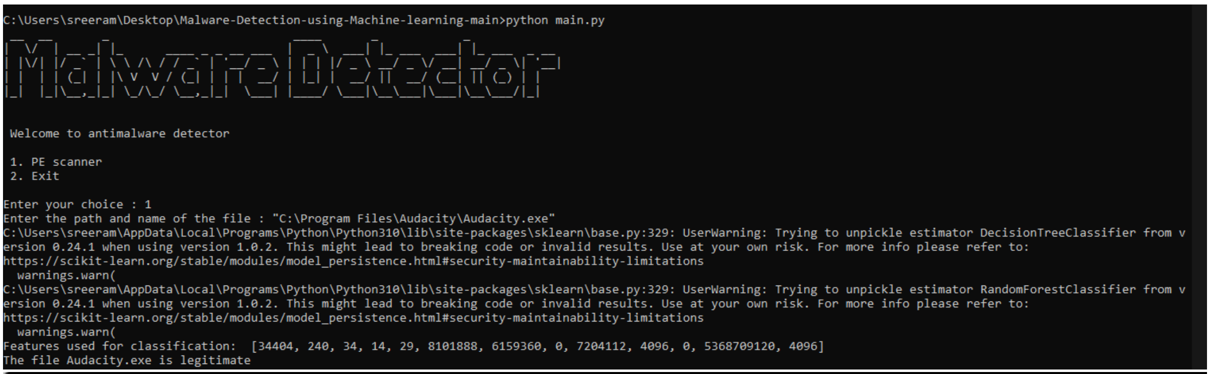
```
res['SizeOfHeapCommit'] = pe.OPTIONAL_HEADER.SizeOfHeapCommit
res['LoaderFlags'] = pe.OPTIONAL_HEADER.LoaderFlags
res['NumberOfRvaAndSizes'] = pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

# Sections
res['SectionsNb'] = len(pe.sections)
entropy = list(map(lambda x:x.get_entropy(), pe.sections))
res['SectionsMeanEntropy'] = sum(entropy)/float(len((entropy)))
res['SectionsMinEntropy'] = min(entropy)
res['SectionsMaxEntropy'] = max(entropy)
raw_sizes = list(map(lambda x:x.SizeOfRawData, pe.sections))
res['SectionsMeanRawsize'] = sum(raw_sizes)/float(len((raw_sizes)))
res['SectionsMinRawsize'] = min(raw_sizes)
res['SectionsMaxRawsize'] = max(raw_sizes)
virtual_sizes = list(map(lambda x:x.Misc_VirtualSize, pe.sections))
res['SectionsMeanVirtualsize'] = sum(virtual_sizes)/float(len(virtual_sizes))
res['SectionsMinVirtualsize'] = min(virtual_sizes)
res['SectionMaxVirtualsize'] = max(virtual_sizes)

#Imports
try:
    res['ImportsNbDLL'] = len(pe.DIRECTORY_ENTRY_IMPORT)
    imports = sum([x.imports for x in pe.DIRECTORY_ENTRY_IMPORT], [])
    res['ImportsNb'] = len(imports)
    res['ImportsNbOrdinal'] = 0
except AttributeError:
    res['ImportsNbDLL'] = 0
    res['ImportsNb'] = 0
    res['ImportsNbOrdinal'] = 0

#Exports
```

```
try:
    res['ExportNb'] = len(pe.DIRECTORY_ENTRY_EXPORT.symbols)
except AttributeError:
    # No export
    res['ExportNb'] = 0
#Resources
resources= get_resources(pe)
res['ResourcesNb'] = len(resources)
if len(resources)> 0:
    entropy = list(map(lambda x:x[0], resources))
    res['ResourcesMeanEntropy'] = sum(entropy)/float(len(entropy))
    res['ResourcesMinEntropy'] = min(entropy)
    res['ResourcesMaxEntropy'] = max(entropy)
    sizes = list(map(lambda x:x[1], resources))
    res['ResourcesMeanSize'] = sum(sizes)/float(len(sizes))
    res['ResourcesMinSize'] = min(sizes)
    res['ResourcesMaxSize'] = max(sizes)
else:
    res['ResourcesNb'] = 0
    res['ResourcesMeanEntropy'] = 0
    res['ResourcesMinEntropy'] = 0
    res['ResourcesMaxEntropy'] = 0
    res['ResourcesMeanSize'] = 0
    res['ResourcesMinSize'] = 0
    res['ResourcesMaxSize'] = 0

# Load configuration size
try:
    res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
except AttributeError:
    res['LoadConfigurationSize'] = 0
```

```
    # Version configuration size
    try:
        version_infos = get_version_info(pe)
        res['VersionInformationSize'] = len(version_infos.keys())
    except AttributeError:
        res['VersionInformationSize'] = 0
    return res


if __name__ == '__main__':

    #Loading the classifier.pkl and features.pkl
    clf = joblib.load(r"C:\Users\sreeram\Desktop\Malware-Detection-using-Machine-learning-main\Classifier\classifier.pkl")
    features = pickle.loads(open(os.path.join(r"C:\Users\sreeram\Desktop\Malware-Detection-using-Machine-learning-main\Classifier\features.pkl"),'rb').read())

    #extracting features from the PE file mentioned in the argument
    data = extract_infos(sys.argv[1])

    #matching it with the features saved in features.pkl
    pe_features = list(map(lambda x:data[x], features))
    print("Features used for classification: ", pe_features)

    #prediciting if the PE is malicious or not based on the extracted features
    res= clf.predict([pe_features])[0]
    print ('The file %s is %s' % (os.path.basename(sys.argv[1]),['malicious', 'legitimate'][res]))
```

Results

Testing a file whether it is malicious or legitimate



```
Do you want to search again? (y/n)y

 Welcome to antimalware detector

 1. PE scanner
 2. Exit

Enter your choice : 1
Enter the path and name of the file : "C:\Users\sreeram\Desktop\Malware-Detection-using-Machine-learning-main
\Google_Adobe_FlashPlayer.exe"
C:\Users\sreeram\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning:
Trying to unpickle estimator DecisionTreeClassifier from version 0.24.1 when using version 1.0.2. This might
lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\sreeram\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:329: UserWarning:
Trying to unpickle estimator RandomForestClassifier from version 0.24.1 when using version 1.0.2. This might
lead to breaking code or invalid results. Use at your own risk. For more info please refer to:
https://scikit-learn.org/stable/modules/model_persistence.html#security-maintainability-limitations
  warnings.warn(
Features used for classification:  [332, 224, 34, 8, 0, 7168, 2048, 0, 14982, 8192, 16384, 4194304, 8192]
The file Google_Adobe_FlashPlayer.exe is malicious
Do you want to search again? (y/n)
```

Model1:

A Machine Learning approach for Malware Detection

Importing all the required libraries

```python
import os
import pandas
import numpy
import pickle
import sklearn.ensemble as ek
from sklearn.model_selection import train_test_split
from sklearn import tree, linear_model
from sklearn.feature_selection import SelectFromModel
import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn import svm
from sklearn.linear_model import LinearRegression
```

Loading the initial dataset delimited by |

```python
dataset = pandas.read_csv('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/data.csv',sep='|', low_memory=False
```

```python
dataset.head()
```

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	115712
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	19968
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	621568
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	369152
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	247296

5 rows × 57 columns

```
dataset.describe()
```

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUninitializedData	AddressOfEr
count	138047.000000	138047.000000	138047.000000	138047.000000	138047.000000	1.380470e+05	1.380470e+05	1.380470e+05	1.380
mean	4259.069274	225.845632	4444.145994	8.619774	3.819286	2.425956e+05	4.504867e+05	1.009525e+05	1.719
std	10880.347245	5.121399	8186.782524	4.088757	11.862675	5.754485e+06	2.101599e+07	1.635288e+07	3.430
min	332.000000	224.000000	2.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000
25%	332.000000	224.000000	258.000000	8.000000	0.000000	3.020800e+04	2.457600e+04	0.000000e+00	1.272
50%	332.000000	224.000000	258.000000	9.000000	0.000000	1.136640e+05	2.631680e+05	0.000000e+00	5.288
75%	332.000000	224.000000	8226.000000	10.000000	0.000000	1.203200e+05	3.850240e+05	0.000000e+00	6.157
max	34404.000000	352.000000	49551.000000	255.000000	255.000000	1.818587e+09	4.294966e+09	4.294941e+09	1.074

8 rows × 55 columns

Number of malicious files vs Legitimate files in the training set

```
dataset.groupby(dataset['legitimate']).size()
```

```
legitimate
0    96724
1    41323
dtype: int64
```

Dropping columns like Name of the file, MD5 (message digest) and label

```
X = dataset.drop(['Name','md5','legitimate'],axis=1).values
y = dataset['legitimate'].values
```

ExtraTreesClassifier

ExtraTreesClassifier fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting

```
extratrees = ek.ExtraTreesClassifier().fit(X,y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
```

ExtraTreesClassifier helps in selecting the required features useful for classifying a file as either Malicious or Legitimate

14 features are identified as required by ExtraTreesClassifier

```
nbfeatures
```

```
14
```

Cross validation is applied to divide the dataset into random train and test subsets. test_size = 0.2 represent the proportion of the dataset to include in the test split

```
X_train, X_test, y_train, y_test = train_test_split(X_new, y ,test_size=0.2)
```

```
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
```

The features identified by ExtraTreesClassifier

```python
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]], extratrees.feature_importances_[index[f]]))
    features.append(dataset.columns[2+f])
```

```
1. feature DllCharacteristics (0.125561)
2. feature Machine (0.113050)
3. feature Characteristics (0.109813)
4. feature VersionInformationSize (0.085126)
5. feature ImageBase (0.062561)
6. feature Subsystem (0.062412)
7. feature SectionsMaxEntropy (0.054795)
8. feature ResourcesMaxEntropy (0.050192)
9. feature MajorSubsystemVersion (0.039885)
10. feature SizeOfOptionalHeader (0.034062)
11. feature ResourcesMinEntropy (0.026107)
12. feature SectionsMinEntropy (0.025405)
13. feature MajorOperatingSystemVersion (0.024047)
14. feature SizeOfStackReserve (0.023437)
```

Building the below Machine Learning model

```python
model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
        "RandomForest":ek.RandomForestClassifier(n_estimators=50),
        "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
        "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
        "GNB":GaussianNB(),
        "LinearRegression":LinearRegression()
}
```

Training each of the model with the X_train and testing with X_test. The model with best accuracy will be ranked as winner

```python
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score
```

```
DecisionTree : 0.9904744657732706
RandomForest : 0.9942049981890619
Adaboost : 0.9859833393697935
GradientBoosting : 0.988518652662079
GNB : 0.7015574067366896
LinearRegression : 0.6023759034713998
```

```python
winner = max(results, key=results.get)
```

Saving the model

```python
joblib.dump(model[winner],'E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/classifier.pkl')
```

```
['E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/classifier.pkl']
```

```python
open('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/features.pkl', 'wb').write(pickle.dumps(features))
```

```
267
```

Calculating the False positive and negative on the dataset

```python
clf = model[winner]
res = clf.predict(X_new)
mt = confusion_matrix(y, res)
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 : 0.096150 %
False negative rate : 0.193597 %
```

```python
# Load classifier
clf = joblib.load('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/classifier.pkl')
#load features
features = pickle.loads(open(os.path.join('E:/6th Sem/ISS/Malware Detection project/Machine-Learning-approach-for-Malware-Detection-master/features.pkl'),'rb'
```

Model2:

Users > sreeram > Desktop > Malware-Detection-using-Machine-learning-main > ML_Model > PE.ipynb > M↓Loading the classifier and features > # Load classifier⏎clf = joblib.load('classifier.pkl')⏎#load features⏎features

+ Code + Markdown | Run All Clear Outputs of All Cells | Outline ··· Select Kernel

Importing the dataset

Source :

```python
import pandas as pd
dataset = pd.read_csv(r"C:\Users\sreeram\Desktop\Malware-Detection-using-Machine-learning-main\Dataset\data.csv", sep='|')
```

Python

About the dataset

+ Code + Markdown

```python
dataset.head()    #Top 5 row of the dataset
```

Python

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	115712
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	19968
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	621568
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	369152
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	247296

5 rows × 57 columns

```python
dataset.tail()     #Last 5 row of the dataset
```

Python

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVe
138042	VirusShare_8e292b418568d6e7b87f2a32aee7074b	8e292b418568d6e7b87f2a32aee7074b	332	224	258	11	
138043	VirusShare_260d9e2258aed4c8a3bbd703ec895822	260d9e2258aed4c8a3bbd703ec895822	332	224	33167	2	
138044	VirusShare_8d088a51b7d225c9f5d11d239791ec3f	8d088a51b7d225c9f5d11d239791ec3f	332	224	258	10	
138045	VirusShare_4286dccf67ca220fe67635388229a9f3	4286dccf67ca220fe67635388229a9f3	332	224	33166	2	
138046	VirusShare_d7648eae45f09b3adb75127f43be6d11	d7648eae45f09b3adb75127f43be6d11	332	224	258	11	

5 rows × 57 columns

```python
dataset.columns    # name of the columns
```

Python

```
Index(['Name', 'md5', 'Machine', 'SizeOfOptionalHeader', 'Characteristics',
       'MajorLinkerVersion', 'MinorLinkerVersion', 'SizeOfCode',
       'SizeOfInitializedData', 'SizeOfUninitializedData',
       'AddressOfEntryPoint', 'BaseOfCode', 'BaseOfData', 'ImageBase',
       'SectionAlignment', 'FileAlignment', 'MajorOperatingSystemVersion',
       'MinorOperatingSystemVersion', 'MajorImageVersion', 'MinorImageVersion',
       'MajorSubsystemVersion', 'MinorSubsystemVersion', 'SizeOfImage',
       'SizeOfHeaders', 'CheckSum', 'Subsystem', 'DllCharacteristics',
       'SizeOfStackReserve', 'SizeOfStackCommit', 'SizeOfHeapReserve',
       'SizeOfHeapCommit', 'LoaderFlags', 'NumberOfRvaAndSizes', 'SectionsNb',
       'SectionsMeanEntropy', 'SectionsMinEntropy', 'SectionsMaxEntropy',
       'SectionsMeanRawsize', 'SectionsMinRawsize', 'SectionMaxRawsize',
       'SectionsMeanVirtualsize', 'SectionsMinVirtualsize',
       'SectionMaxVirtualsize', 'ImportsNbDLL', 'ImportsNb',
       'ImportsNbOrdinal', 'ExportNb', 'ResourcesNb', 'ResourcesMeanEntropy',
       'ResourcesMinEntropy', 'ResourcesMaxEntropy', 'ResourcesMeanSize',
       'ResourcesMinSize', 'ResourcesMaxSize', 'LoadConfigurationSize',
       'VersionInformationSize', 'legitimate'],
      dtype='object')
```

```python
dataset.describe(include="all")    # summary of numeric attributes
```

Python

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfIniti
count	138047	138047	138047.000000	138047.000000	138047.000000	138047.000000	138047.000000	1.380470e+05	1.3
unique	107488	138047	NaN	NaN	NaN	NaN	NaN	NaN	
top	mshtml.dll	631ea355665f28d4707448e442fbf5b8	NaN	NaN	NaN	NaN	NaN	NaN	
freq	187	1	NaN	NaN	NaN	NaN	NaN	NaN	
mean	NaN	NaN	4259.069274	225.845632	4444.145994	8.619774	3.819286	2.425956e+05	4.5
std	NaN	NaN	10880.347245	5.121399	8186.782524	4.088757	11.862675	5.754485e+06	2.1
min	NaN	NaN	332.000000	224.000000	2.000000	0.000000	0.000000	0.000000e+00	0.0
25%	NaN	NaN	332.000000	224.000000	258.000000	8.000000	0.000000	3.020800e+04	2.4
50%	NaN	NaN	332.000000	224.000000	258.000000	9.000000	0.000000	1.136640e+05	2.6
75%	NaN	NaN	332.000000	224.000000	8226.000000	10.000000	0.000000	1.203200e+05	3.8
max	NaN	NaN	34404.000000	352.000000	49551.000000	255.000000	255.000000	1.818587e+09	4.2

11 rows × 57 columns

```python
dataset.info()    # info about the whole dataset
```

Python

```
Output exceeds the size limit. Open the full output data in a text editor
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138047 entries, 0 to 138046
Data columns (total 57 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   Name                        138047 non-null  object
 1   md5                         138047 non-null  object
 2   Machine                     138047 non-null  int64
 3   SizeOfOptionalHeader        138047 non-null  int64
 4   Characteristics             138047 non-null  int64
 5   MajorLinkerVersion          138047 non-null  int64
 6   MinorLinkerVersion          138047 non-null  int64
 7   SizeOfCode                  138047 non-null  int64
 8   SizeOfInitializedData       138047 non-null  int64
 9   SizeOfUninitializedData     138047 non-null  int64
 10  AddressOfEntryPoint         138047 non-null  int64
 11  BaseOfCode                  138047 non-null  int64
 12  BaseOfData                  138047 non-null  int64
 13  ImageBase                   138047 non-null  float64
 14  SectionAlignment            138047 non-null  int64
 15  FileAlignment               138047 non-null  int64
 16  MajorOperatingSystemVersion 138047 non-null  int64
 17  MinorOperatingSystemVersion 138047 non-null  int64
```

```python
dataset["legitimate"].value_counts()    # count of malware (0) and benign (1) files in dataset
```

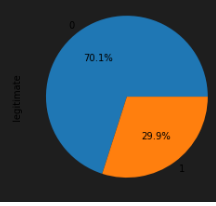
Python

```
0    96724
1    41323
Name: legitimate, dtype: int64
```

Visualization

```python
import matplotlib.pyplot as plt

dataset["legitimate"].value_counts().plot(kind="pie",autopct="%1.1f%%")
plt.show()
```

Python



```python
import os
import pandas
import numpy
import pickle
import pefile
import sklearn.ensemble as ek
from sklearn.feature_selection import SelectFromModel
import joblib
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression
from sklearn.metrics import confusion_matrix
from sklearn import svm
import sklearn.metrics as metrics
```

Python

Feature Selection

```python
# Feature
X = dataset.drop(['Name','md5','legitimate'],axis=1).values    #Droping this because classification model will not accept object type elements (float and int
# Target variable
y = dataset['legitimate'].values
```

Python

Data Fitting and choosing the important variables

```python
extratrees = ek.ExtraTreesClassifier().fit(X,y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
```

Python

```python
#Number of important features
nbfeatures
```

Python

```
13
```

```python
#splitting the data (70% - training and 30% - testing)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_new, y ,test_size=0.29, stratify = y)
```

Python

```python
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
```

Python

All the required features

```python
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]], extratrees.feature_importances_[index[f]]))
    features.append(dataset.columns[2+f])
```

Python

```
1. feature DllCharacteristics (0.147093)
2. feature Machine (0.109596)
3. feature Characteristics (0.086385)
4. feature VersionInformationSize (0.074387)
5. feature SectionsMaxEntropy (0.074359)
6. feature ResourcesMaxEntropy (0.055664)
7. feature MajorSubsystemVersion (0.052161)
8. feature Subsystem (0.050708)
9. feature ImageBase (0.045204)
10. feature ResourcesMinEntropy (0.037663)
11. feature SizeOfOptionalHeader (0.037558)
12. feature MajorOperatingSystemVersion (0.024381)
13. feature SectionsMeanEntropy (0.019135)
```

Testing which Classifier will give better result

```python
model = { "DecisionTree": DecisionTreeClassifier(max_depth=10),
          "RandomForest":ek.RandomForestClassifier(n_estimators=50),
          "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
          "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
          "GNB":GaussianNB(),
          "LinearRegression":LinearRegression(),
          }
```

Python

```python
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score
```

Python

```
DecisionTree : 0.9899085777089474
RandomForest : 0.9933306689314083
Adaboost : 0.9859869111255433
GradientBoosting : 0.9871359344557127
GNB : 0.7006794224908828
LinearRegression : 0.5262290467642269
```

```python
winner = max(results, key=results.get)# Selecting the classifier with good result
print("Using", winner, "for classification, with",len(features), 'features.')
```

Python

```
Using RandomForest for classification, with 13 features.
```

Saving the machine as classifier.pkl and features to be extracted as features.pkl

```python
joblib.dump(model[winner],'classifier.pkl')
open('features.pkl', 'wb').write(pickle.dumps(features))
```

Python

```
251
```

Loading the classifier and features

```python
# Load classifier
clf = joblib.load('classifier.pkl')
#load features
features = pickle.loads(open(os.path.join('features.pkl'),'rb').read())
```

Python