Malware detection using machine learning

Fan Li
CSI-5500 Operating Systems
School of Engineering and Computer Science Oakland University
fanli@oakland.edu

Abstract

In recent years, people have become increasingly inseparable from the Internet in their working and lives, which has led to the existence of underground economy such as malware. Malware can steal our information and cause damage to individuals or businesses. Compared with firewalls, data encryption and other traditional network security protection methods, intrusion detection and malware detection systems have irreplaceable advantages in information misinformation, software leakage, and have become the main approaches of protecting network security. In this paper, I tried to detect malware using machine learning. I extracted static features from binary files which will be used as the dataset for training the model. Comparing the accuracies of different models on different datasets and finally get the best accuracy is 99.79%.

1 Introduction

Malware threats have been with us since the development of computer technology. Malwares have also been iteratively updated with the times, and the number of global cybercrimes and security breaches has increased in recent years. Malwares often steal sensitive information (such as credit card numbers or passwords) from users without their knowledge, or steal their email accounts to send fake emails, thereby damaging users' or business property. Malwares include, but is not limited to, viruses, worms, spywares, adwares, and Trojan horses. In order to better protect the security of information property of individuals and enterprises, I proposed a method of utilizing machine learning techniques to detect the malwares. In this study, I have read the relevant literature and in Section II I present the solutions for malware in recent years. Section III shows the types of malwares and the methods for extracting malware features, which are static feature extraction and dynamic feature extraction. In the experiments I used static analysis to extract features from PE files. In the model selection, I choose several commonly used models, including GaussianNB, MLP, Linear Regression, Decision Tree, and Gradient Boosting. I design three different data preprocessing methods to train the model, the first one retains the extracted feature information, the second one uses feature selection to process the data set and filters out some features before training the model, and the third one uses an AutoEncoder to encode original features to their latent representations. Finally, conclusions are drawn by comparing the performance of different models on different data. I concluded that DecisionTree has the highest accuracy, the accuracy is 99.79%.

2 Related work

A framework called PEMiner is proposed in [1] for malware detection and classification. They extracted several features from PE files: the COFF header, the optional header and some section attributes. In [2], a new approach to monitor malware called Malware Instruction Set (MIST), which uses data mining and machine learning techniques to achieve effective software behavior analysis. A static analysis approach is used in [3] to extract three different features: strings, system resource information, and byte sequences. In [4], n-grams of bytecodes are used as features and machine learning algorithms such as: decision trees [4], naive Bayes, support vector machines and boosting are applied, where boosted decision trees outperform the other methods. In [5], the authors categorize methods according to

- *W*hat is the problem they want to solve.
- *W*hat are the types of features extracted from the portable files.
- *W*hat are the machine learning algorithms they use.

In addition, feature selection can be used with self-supervised learning to improve performance. In this research, I will deal with the malware detect using machine learning from two aspects: obtaining the malware features

through feature extraction, then training the model with the features.

3 Background

3.1 Malware

Malware is a malicious application or code that damages or disrupts the normal use of an endpoint device. When a device is infected with malware, you may experience unauthorized access, data leakage, or the device may be locked until you pay a ransom. There are three main types of malwares on the Web today: viruses, worms, and Trojans, and five widely used types of malwares: rootkits, keyloggers, ransomware, spyware, and adware [6].
Virus: A virus is a type of malware, but not all malware is a virus. Most malware programs are not viruses, and computer viruses try to attach themselves to legitimate programs in order to spread their malicious code to as many other programs and devices as possible. When the execution of the victim file is activated, the virus is also executed and replicates to infect other files on the system.
Typically viruses are spread by exploiting vulnerabilities in software applications and operating systems, they can also infect user devices through

- *I*nfected email attachments
- *M*alicious scripts and website downloads
- *R*emovable external media

Nowadays, pure computer viruses are not very common, which is considered a good thing for users, but viruses are very difficult to remove because they execute malware from legitimate programs, so in most cases the only way to remove a virus is to quarantine or delete the infected file. Worms: Like viruses, worms are a type of malware that can also spread from the host computer to other devices. However, unlike viruses, worms are more insidious because they do not require human intervention to spread and are runnable, standalone malicious programs. Based on their unique characteristics of self-replication, worms therefore spread faster and more widely and are more destructive than viruses. Trojan horse: A Trojan horse is a malicious software or code program used by hackers to remotely control a device. It is a piece of malicious code with special functions hidden in a normal program that can perform illegal operations such as monitoring, data modification, and stealing important information from the controlled computer. Trojan horses are usually disguised and hidden in the following legitimate files: Office documents (e.g. Word and Excel, PDF files, executable files. Trojans are usually spread via email or pushed when users visit infected websites. Rootkit: Rootkits are a remote access tool or application provided for use by remote administrators of devices. Hackers can use Rootkits to hide traces of intrusion activity, retain ROOT access, and also hide malicious programs in the operating system. Spyware: Spyware is a type of surveillance software that allows someone to observe your activities without your knowledge or authorization, and through this type of software can access your sensitive personal information and data.

3.2 Malware analysis

3.2.1 PE file format

The malware used in this paper is PE files. PE files are the general name of executable files under Windows, such as DLL, EXE, OCX, SYS, etc. Generally speaking, the structure of PE files is shown in the following diagram: DOS header, NT header, section table and specific sections in order from the starting position [5]. The DOS header is used for MS-DOS compatibility to indicate a text when the file runs on MS-DOS, in most cases: This program cannot be run in DOS mode. Another purpose is to indicate the location of the NT header in the file. The NT header contains the main information about the windows PE file, including a 'PE' signature, the PE file header (IMAGE_FILE_HEADER) and the PE optional header (IMAGE_OPTIONAL_HEADER32). section table: is the description of subsequent sections of the PE file, windows loads each section according to the description of the section table. section: each section is actually a container that can contain code, data, etc. Each section can have independent memory permissions, for example, the code section has read/execute permissions by default, and the name and number of sections can be defined by yourself, not necessarily the three in the bellow figure 1.

3.2.2 PE file execution sequence

When a PE file is executed, the PE loader first checks the offset of the PE header in the DOS header. If it finds it, it jumps directly to the PE header location.
After the PE loader jumps to the PE header, the second step is to check if the PE header is valid. If the PE header is valid, it jumps to the end of the PE header.
After the PE loader executes the second step, it starts to read the section information in the section table and uses file mapping (when executing a PE file, Windows does not read the whole file into memory at the beginning, but uses the memory mapping mechanism, that is, the Windows loader only establishes the virtual address and the PE file when it loads, and only when it actually executes the PE file does the file mapping mechanism work. This means that the Windows loader only establishes the mapping relationship

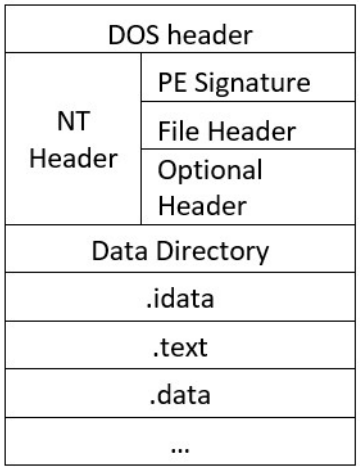

Figure 1. PE format

between the virtual address and the PE file at the time of loading, and the page is committed from disk to physical memory only when the instruction in a memory page is actually executed or when the data in a page is accessed; this mechanism makes the speed of file loading not much related to the file size) method of mapping these sections to memory, and attaching the read and write attributes of the specified sections in the section table. After the PE file is mapped into memory, the PE loader will continue to process the logical part of the PE file that resembles an import table.

3.2.3 Static analysis

There are two basic approaches to malware analysis: (1) static analysis and (2) dynamic analysis. On the one hand, static analysis involves checking the malware without running it. On the other hand, dynamic analysis involves running the malware [7]. Static analysis means examining the code or structure of an executable file without executing it. This method can protect the security of the operating system and also check out whether the file is malicious or not. Common static extraction methods are

- *A*nalyzing PE file headers and sections. PE files contain metadata about the file itself, such as the actual section of the file.
- *C*ollecting metadata about the executable's linked libraries and functions, as well as the files contained in the header files. This data provides information about the code base and functions that are common to many programs. The names of these Windows functions can give us an idea of what the executable is.
- *F*inding sequences of characters or strings. Searching for a program's strings is the easiest way to get hints about its functions. Strings extracted from binary files can contain references to file paths modified or accessed by the executable, URLs accessed by the program, domain names, IP addresses, attack commands, names of loaded Windows dynamic link libraries (dlls), registry entries, etc. Utility Tools String 5 can be used to search for ASCII or Unicode strings while ignoring context and formatting in executable files.
- *S*earch for packaged/encrypted code. Malware writers often use packaging and encryption to make their files more difficult to analyze. Packaged or encrypted software programs usually contain few strings and higher entropy than legitimate programs.
- *T*ranslation of machine code into assembly language

3.2.4 Dynamic analysis

Dynamic features are features extracted from the execution of the malware at runtime. Dynamic analysis involves monitoring the malware while it is running (and observing the actual sequence of instructions executed or the sequence of API functions triggered), or examining the system after the malware has been executed. The creation and modification of files and registry values. Generally speaking, dynamic features include the following aspects: operation behavior of files; operation behavior of registry keys; loading behavior of dynamic link libraries; operation behavior of process access; system service behavior; network access requests; and API calls. Compared with static analysis, dynamic analysis is more complex and time-consuming, and uses relatively higher-level features, and is therefore less interpretable.
Example of dynamic feature design, the article was published in 2016 [8] and the PE file was entered directly into the Cuckoo sandbox, which is an open source sandbox, and is more general for extracting dynamic features in academic papers; then a preprocessing operation was performed to convert the text into a vector representation, for example, 200 dynamic features were extracted, which can be represented using a 200-dimensional vector, and the position of each array represents the corresponding API, and then The resulting sequence is fed into the convolutional neural network LSTM for classification, and finally the structure of the family classification is obtained.

4 Approach

The main process of this project is shown in Figure 4, which consists of three steps [9]: input data, feature extraction, and model training. In this project, I am doing static

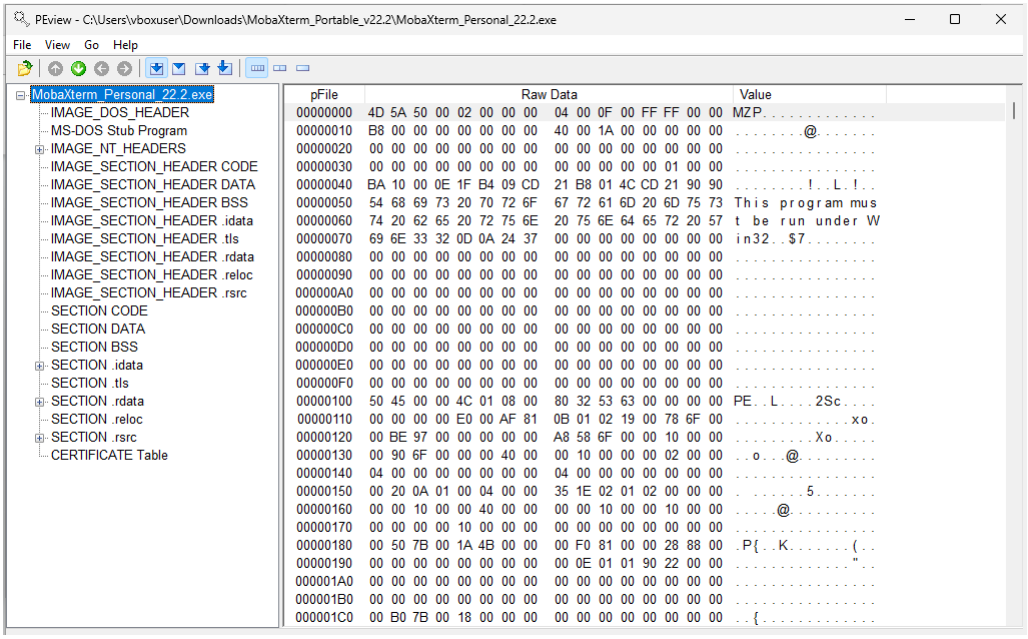

Figure 2. PE file

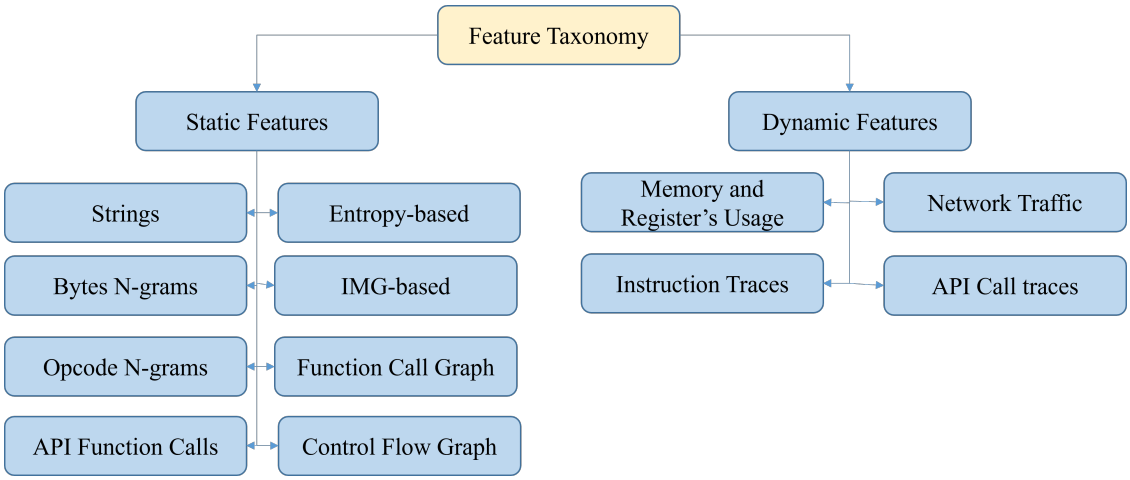

Figure 3. Feature Taxonomy

malware analysis. After downloading the datasets, I am going to extract the features from PE header. The features will be stored csv file and used for training the models.

4.1 Extract features

I extracted the static features from PE files using the pefile library. The following I will introduce some features that have been extract.
Entropy: entropy measures the randomness of the data in a file and is used to determine whether a file contains hidden data or suspicious scripts. Many malware programs use entropy as a packing technique [10]. Packaging leads to an increase in entropy. So entropy can be used as an indicator of malware programs. Also, some benign programs use packing. In general, a higher percentage of malware programs use packing.
Checksum: The algorithm for computing the checksum is incorporated into IMAGHELP.DLL. The following are checked for validation at load time: all drivers, any DLL loaded at boot time, and any DLL that is loaded into a critical Windows process. According to the studies, it has been observed that the value of this field is zero for many malware programs, while benign programs have significantly larger values for this field.
FileInfo: VS_FIXEDFILEINFO Contains version information for a file. FileVersionLS and ProductVersionLS will contains the version information about the binary file. Each file has a different number of these fields. Since benign programs are valid programs produced by reliable individuals or companies, they usually contain these fields, while many malware programs lack this set or have only a small number of these fields.

4.2 Preprocessing data

There are many redundant features are included after extracting features. It is necessary to select features that are useful for distinguishing software types.
In this step, I choose three ways to preprocessing the data: only preprocessing data simple, using feature selection to filter out some features, and using AutoEncoder to process the data. Dataset One, only simple data processing, only need to remove the information in the file that is not related to the training model, such as file name, file md5 value, etc. Dataset two, the purpose of feature selection is to select the most relevant set of features among these features, while leaving fewer features to improve the classification performance. In this step, I used the Extra Trees Classifier to select the features.
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 it's 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.
To perform feature selection using the above forest structure, during the construction of the forest, for each feature, the normalized total reduction in the mathematical criteria used in the decision of feature of split (Gini Index if the Gini Index is used in the construction of the forest) is computed. This value is called the Gini Importance of the feature. To perform feature selection, each feature is ordered in descending order according to the Gini Importance of each feature and the user selects the top k features according to his/her choice. Dataset three, use AutoEncoder to process the data and map from low-dimensional data to high-dimensional data. An autoencoder is a type of artificial neural network used to learn efficient coding of unlabeled data (unsupervised learning) [3]. The encoding is validated and refined by attempting to regenerate the input from the encoding. The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data ("noise"). But this project, I used it to ascend dimension. This way may contain hidden information that does not appear in the original features.

4.3 Training models

I choose 5 models in this paper, including GaussianNB, MLP, LinearRegression, DecisionTree, and GradientBoosting. The dataset is generally divided into two parts for the whole dataset, one for training and one for validation. In this paper I choose 10-fold cross-validation to handle the training and test sets. 10-fold cross-validation randomly divides the original data into 10 parts. Then chooses one of them as the testing set and the other nine as the training set each time. Then training the model 10 times. In this way, each subset has a chance to be the training set or the test set. In addition, the five models will be trained on three different datasets, and finally the results on the different datasets will be compared.
Gaussian Naive Bayes (GNB) is a classification technique for machine learning based on probabilistic methods and Gaussian distributions. Plain Bayes assumes that each parameter (also called a feature or predictor variable) has the independent ability to predict the output variable. The com-

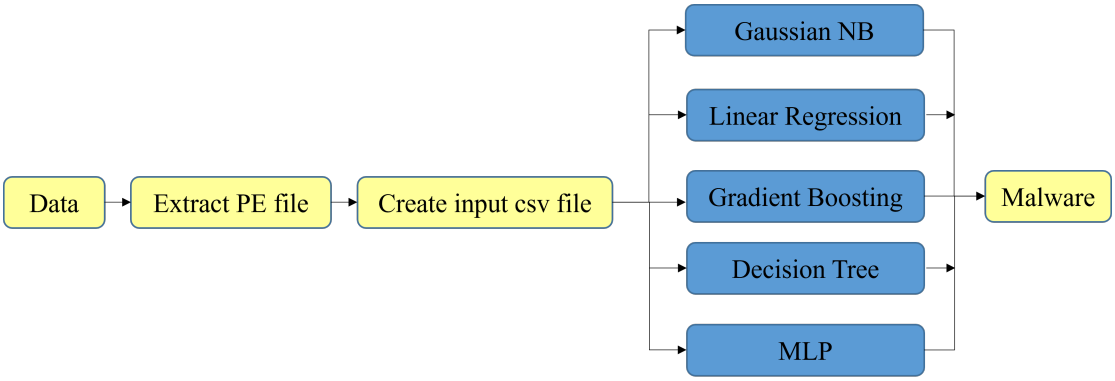

Figure 4. Framework

bination of predictions for all parameters is the final prediction, which returns the probability that the dependent variable will be classified into each group, with the final classification being assigned to the group (class) with the higher probability.

Linear Regression is a linear model for prediction by linear combination of attributes, which aims to find a straight line or a plane or a higher dimensional hyperplane that minimizes the error between the predicted and true values.

Decision tree is a decision analysis method that evaluates the project risk and judges its feasibility by forming a decision tree to find the probability that the expected value of the net present value is greater than or equal to zero on the basis of the known probability of occurrence of various situations, and is a graphical method that intuitively uses probability analysis. Since this decision branch is drawn as a graph resembling the branches of a tree, it is called a decision tree. Decision trees are a basic classification and regression method, and learning usually involves three steps: feature selection, decision tree generation, and decision tree pruning. In machine learning, a decision tree is a predictive model that represents a mapping relationship between object attributes and object values.

The basic idea of Gradient Boosting is to serially generate multiple weak learners, each with the goal of fitting the negative gradient of the loss function of the previously accumulated model so that the cumulative model loss after adding the weak learner is reduced in the direction of the negative gradient. Gradient Boosting can also be understood as gradient descent over the function space. The more familiar gradient descent is usually a gradient descent of values over the parameter space (e.g. training a neural network, where the gradient of the current loss with respect to the parameters is computed in each iteration and the parameters are updated). In Gradient Boosting, on the other hand, a weak learner is generated in each iteration, which fits the gradient of the loss function with respect to the previous cumulative model, and then this weak learner is added to the cumulative model to gradually reduce the loss of the cumulative model. That is, the gradient descent in parameter space uses the gradient information to adjust the parameters to reduce the loss, and the gradient descent in function space uses the gradient to fit a new function to reduce the loss.

MLP consists of three layers: input layer, hidden layer, and output layer, the first layer is called the input layer, the last layer is called the output layer, and the middle layer is called the hidden layer. The number of hidden layers can be chosen according to the respective needs. MLP has good recognition rate and fast classification, but in exchange its training process is time consuming.

5 Experiment

5.1 Project operating environment

- *L*anguage: Python(3.8)
- *P*latform: Jupyter Notebook
- *L*ibraries: Scikit Learn, Pandas, Pickle, Numpy, torch, dfencoder, etc

5.2 Data collection

The malware dataset is obtained from a publicly available website with the address https://virusshare.com/ and the benign software come from https://github.com/iosifache/DikeDataset. The dataset consists of 207781 data entries listing PE information from various sources. 113329 are malicious while 94452 are legitimate. The two figures shown below are the malware downloaded from the virusshare website, the benign software downloaded from github.

3686c09b2a7755811095196bd155757b5316883d27266ca2d808ba38b410163a.exe	6e301f788993268ad839c6e8b251d6b0ee14bddbc9a655bb7e4474524a6d8994.exe
373d0778d083a04943032826a3571e5e8d450a9535f5a901db43c791200a774d.exe	6e4849e0d92ae82380bf4bb6773a1ff27863038838a021ccade723e2adab5d73.exe
3777466cacecedcab6f2e7adc82cfc5f18c0c7f41bea1c68192c4027d53c0d0e.exe	6ec6507c9d9332e7d98f7ba4c14d8fa326e581894e266c72841cd4c2dc894bbc.exe
377c0e34938c5c87ce613c5f2ea64c702baa55d2368c20ab375ecc8b605d0a76.exe	6ee4dca852ae3b8f2fa3dba67bf58ea089d1ad5af0946dad3350339877cb7f29.exe
37ad74048035735bda661eec2bf47e117c9819698c73546f4ab4eae78fe660ac.exe	6f32f8ac6b79b7686cf837194943a424ebf61f67e9fece3aacb2f7ea4f45d6aa.exe
37df0036b3a3d49c4d2d717ae8cff0576811b9a2726fa3b044af5934a48f48f9.exe	6faf6bc7caf49873d2c96592db1eb6fd65a19b1deae414640c1f5de4bdad136b.exe
37df607966b7219faf58bb94e931b455b27e86c52c54878307c9a4c564067cf4.exe	70985dabcf2497a00eb7fe4c21452bbcf7aa56db01895d130a8dac728b712668.exe
380616b38e867f0d99ef4666f8d551e97ce56e83be29fcd4567e68b95d707d01.exe	70bffc9c81bde425bbd3b2eaf54fb82764d15808f92e9338d2b954a5b912b43b.exe
3846f2ce94d43e9ef6c08cd0abfb5c6c6c993b00d8690bbea40cc525b44cc0a3.exe	71e8b80a35184e5061f69b61f3d86be5a03349c8bd0639c26c3152a02346c6f0.exe
38515483413002d884d7354be2a724117ca9e6b0baecfc9d47292a194996f72b.exe	71f59518cfceceeaa6ba958f34273949565d4176ec94a6f017fe992e09225a37.exe
387d26467015fe41d1a694b0dc9f3a4a52e7e57f4ec0cfc63d636bef1d9f4df7.exe	7230968208169f32697afe22a58dd045687f5c64c5c20b0d0471e4b9aa37c783.exe
392b91d6f81850708be8fe0cd10ba983a50d0ffb51ec40af56008333fc43af89.exe	7297a4cca5635208589bd4e8d7c1b090ac7370f3c0ef6759d7075b150cbb9857.exe
39bbaf0755a1f571d03a1e77a578f59bc5a11635270ad758757abbe55bad8e30.exe	729d140140c3af5b46031f1a7652b159b08721afa80776f17afe73633800afc1.exe
3a001b8c4d0e52362bba32d326158c4fbdbf48cefd49e5b1ac23d77bca98093f.exe	72bc39297e239de515b1384d1df053109eb71567ddc3b94cf7b27ad3a5ad31fe.exe
3a26e84b22e7665a55f0857da23a05e3ef3ed430242b6f7635909981cdbeba94.exe	72c494406edd86560f4cc1ec5736b090df77b2736d94ef79236d3870958a3b18.exe
3a44fa455ada6aede5098355efc22825ac336a3faa18d7695054fcf3a7bbffc0.exe	7320783c6e2941f13cbf3cdc75e649f6d8c1054175aa264693a404cec20f4590.exe
3aeec86e232b09526dc62362f3d37c010bdb962eb31e5a18f2e781aa25423033.exe	734db8a127eba3f56ceacddf8d784fb4c6ff0d9524eaf0d2f75acd698ef9084c.exe
3b120c2b06afd93fc3c84a98c497afff4d93e64168ad809498f86568600b38f3.exe	7371ddedc26db5fb2d9080870a422d8f6236a03133dd74b5f8fc62c437cf1b51.exe
3b812075fee45f45ae2d9801c25eb4cadef412b703a1df9c29060bad61fbb039.exe	73bb2538c8a42b16397858d449d8558d666918a17e272aa74934b2014482a9d0.exe
3bc5906d625e95dfb4744c2baa1e1a5d1b3f73ee663cc149f02ef7eaf23986c0.exe	7481b61d95c45a5ebb6dfc318a8a62f1524f815fb8fccc8859b373570c049538.exe
3be77cde3fe49040e294b6e4b4fceb284fcfb65279fcdcad9dc087d2d4406058.exe	7485ff522f4662ae5c91a5248e82370d9f24bb6e548fe279ce9e9e824c6c4d41.exe
3ca96b622f4af8cfdf1d65b9c1c2ee118bc924753ad95d3376b97c6f08153a54.exe	74c72500841035f639975bc6aa321c4a40e6d2080834d94f522353efe52d79d1.exe
3ccc132059602cbf7dc3acf61a7f83d443a8379a784cc660575858bdb455e6cc.exe	751941b4e09898c31791efeb5f90fc7367c89831d4a98637ed505e40763e287b.exe
3cccf40d984007ee8b64fdb0216efa50e850306531dd3f4ac7710a6dc155351b.exe	755394b623e31d32f900de2f37ba04a0aa9a3c3c57dc34b8f5f4792f18c34196.exe
3ce6f5e9aee5a3f921d1a21144ae5b7b557dbfc7999f282adcc2fc522fc787bb.exe	75618935566046844c2d4270687f88dcfe5b50cd9d670adb7bfb0db1b1070beb.exe
3cefaa966d5e204fe6601160e7ba4d27910437bd1e83b8e0e9f2c4c3de210227.exe	75b7971116bc93808372209737c5f272cde2b5409722dd7a13c5b780f98743ef.exe
3d1ac5f16b486805bfea6e9152cc60251b7e8f4a8580187afef978347e523f46.exe	75d348a3330bc527b2b2ff8a0789f711bd51461126f8df0c0aa1647e9d976809.exe
3d8b868359f623a9441e336d32c360352bf819114052599d3c85ab69e60d1de9.exe	76098cc2969c4a090fba5c863bb3bb52b5ce88bee46da9d46aab0252affa0744.exe

Figure 5. Benign software

VirusShare_1e53e39e4937a8d0d2a1cc39e425cd90	VirusShare_469e00cff4397c2aededa9afa4aa217c
VirusShare_1e55220f8503699d86073001e3f100a1	VirusShare_46cd78311d0b71a01ce3b38bb174922f
VirusShare_1e65891fef0f34d60b94cef8dab16729	VirusShare_46e6c3cbf8a7090d1639da8a01c510c0
VirusShare_1e7b3035288347f3990a9d41c3bd4140	VirusShare_46ff7d9764c068327276a156ed8d1ab1
VirusShare_1e7e4222adbebabdcad960774900d812	VirusShare_4713a0bcb1dc74876ed477cc9e612f26
VirusShare_1e93c9717ca2ec351edd3901b549f5ff	VirusShare_47a8916d95cc9b621319a1fc2d595793
VirusShare_1ea6bfc530a6d94d5bf6da329febb669	VirusShare_47ab049b781345e614b5d65b03e5cf3d
VirusShare_1ecc228feb6806c6da135cbd7aeeabb7	VirusShare_47d1327a4b304bdab90b2346a6b42a1e
VirusShare_1ed0e9ebc500f817a1037f5d5f0babe1	VirusShare_47d1c65b6adbce27f9aee427cd969d7c
VirusShare_1eea3db743a4c08507f1c0689d1625e0	VirusShare_47d4f1956ada44058ebae41475df2600
VirusShare_1f0c2d3ded21338503c8969d3a77fc12	VirusShare_47ea7d3f7061f62a6a0a029d14e2c524
VirusShare_1f1c9591c1d04c9330143e660c1c4fea	VirusShare_485177fa6b5bd809e95a4d9f3de0f3cb
VirusShare_1f4fe55765b4d0781ddb7ab63c44d7fd	VirusShare_485bdd853c7213bd2601c9b831a11f93
VirusShare_1f604c85e196114d3835c1972429ada9	VirusShare_48674592721d6dd9e24b6b5e66d44464
VirusShare_1f68ac0f8c961a1e1be2fa83f7fb8690	VirusShare_4877ba0915afd95d40803e8815858a57
VirusShare_1f9b6539dad6faa210b4683ca0784b83	VirusShare_48f5c0174f51034ac69f771ed280bd9a
VirusShare_1fbad40d75a8b718b559ef97ae2c6184	VirusShare_490bb5822d6114bbdb8282f06fdfe483
VirusShare_1fd4ad60e5d4f572ecebcdd2d1171a73	VirusShare_495354d49c277b0da92b0b1a9d0c50d4
VirusShare_1fd51d59530fd2a63045eac7fa906810	VirusShare_49541c43a2368eea84a0e502cf0ae268
VirusShare_202f19f9f065256e329b8ab3c1372ad6	VirusShare_49560a16fb4b87062058a976fff76832
VirusShare_203c339e9a23f4b8e44a95083573bfad	VirusShare_4961fd5df613ce1af12988f2eb6cc7d0
VirusShare_2043e0420eb3cefec2cf5d1cdb32503d	VirusShare_496be37084afac4664955b657b34ab50
VirusShare_204d324c13352391092693f7be08ab31	VirusShare_49822c629fe40089ce1289bf36a35971
VirusShare_2053d87306cc6954973b4712e27ef604	VirusShare_49d2efdcb8db7caa6d513888a9a14ebd

Figure 6. Malware

5.3 Extract feature

Partial example codes of extracting features are presented in 5.3

There are 57 features after extracting the static features.

```python
def get_version_info(pe):
    """Return version infos"""
    #https://learn.microsoft.com/zh-cn/windows/win32/api/verrsrc/ns-verrsrc-vs_fixedfileinfo
    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
```

Figure 7. Partial example codes of extracting features

5.4 Preprocessing data

In dataset one, there still are 54 features after dropping the name of file, md5 value and legitimate information. In dataset two, there only have 13 features after using feature selection. In dataset three, I can not calculate the features since the data processed by AutoEncoder is uninterpretable data. Figure 5.4 demonstrates features in dataset two. Figure **??** shows the loss curves of training a dfencoder model. Figure 5.4 presents example code of training a dfencoder model.

Out[37]:

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVer
0	572733582ed2076416e035a1d670d12d984c8d56a1b94d...	e0c8ac0427fe71d9c5913fadb5d19d1b	332	224	258	
1	2999b0709f50ed5026039496142254af172990a50e2a6e...	19611ed77cbac092503767eef94fbc50	332	224	783	
2	ae484f4c0341f050d40d3e148be8d11ce86358349ddf31...	4794245bd7eb172176aa65074895fc0f	332	224	290	
3	dec0efefe7749ab45c541f8c2c0bc4fa946e865025e209...	a84e0f438a5637255096c185fb50bb54	34404	240	34	
4	6b8d1a1f785c103e2d75702c22491d1b3942878bcba796...	26a13fbba483291b5e14329fc1f04837	34404	240	34	
...	...	...	...	...	...	
207776	VirusShare_b2c592bb21e0a566e512ff697bd73cef	b2c592bb21e0a566e512ff697bd73cef	332	224	271	
207777	VirusShare_d6ec89f00667d2439391c9a37b8ce49e	d6ec89f00667d2439391c9a37b8ce49e	332	224	271	
207778	VirusShare_9dd3ee41cb1b7e8902e38eeb645a772f	9dd3ee41cb1b7e8902e38eeb645a772f	332	224	33166	
207779	VirusShare_ad480c003f5e93a0306ec3387fdb448d	ad480c003f5e93a0306ec3387fdb448d	332	224	271	
207780	VirusShare_2eb062a323b59e0734ae1f40c46a9608	2eb062a323b59e0734ae1f40c46a9608	332	224	271	

207781 rows × 57 columns

Figure 8. Features after extraction

```
1. feature Subsystem (0.150836)
2. feature SizeOfOptionalHeader (0.140299)
3. feature DllCharacteristics (0.088468)
4. feature Machine (0.085965)
5. feature MajorSubsystemVersion (0.072326)
6. feature SectionsMaxEntropy (0.045958)
7. feature SectionAlignment (0.044875)
8. feature SizeOfHeaders (0.034736)
9. feature LoadConfigurationSize (0.029276)
10. feature MinorSubsystemVersion (0.023359)
11. feature SizeOfStackReserve (0.022674)
12. feature ImageBase (0.020640)
13. feature SectionsNb (0.019799)
```

Figure 9. Features in dataset two

5.5 Training models & result

There are five models used in my experiments:

- *G*aussianNB
- *M*LPClassifier
- *L*inearRegression
- *D*ecisionTree
- *G*radientBoosting

5.5.1 Data set one

The highest accuracy model is DecisionTree, its accuracy is 99.79%.

Model	Accuracy
GaussianNB	52.24%
MLPClassifier	97.03%
LinearRegression	78.39%
DecisionTree	99.79%
LinearRegression	98.24%

5.5.2 Data set two

The highest accuracy model is DecisionTree, its accuracy is 99.52%.

Model	Accuracy
GaussianNB	45.51%
MLPClassifier	80.17%
LinearRegression	74.21%
DecisionTree	99.52%
LinearRegression	98.14%

5.5.3 Data set three

The highest accuracy model is MLP, its accuracy is 99.00%.

Model	Accuracy
GaussianNB	71.49%
MLPClassifier	99.00%
LinearRegression	68.22%
DecisionTree	98.39%
LinearRegression	93.28%

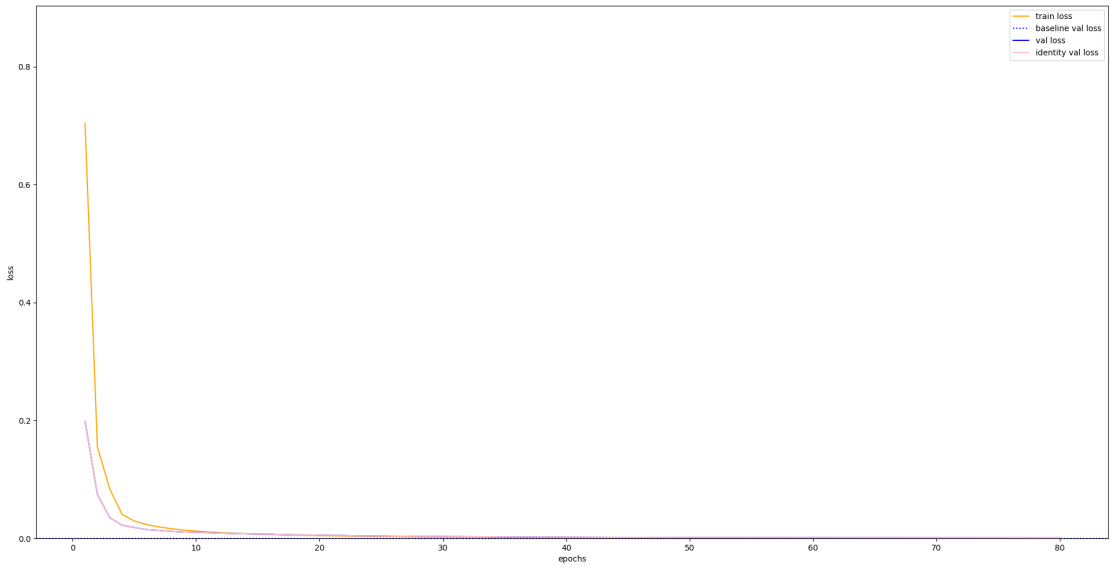

Figure 10. Training loss of dfencoder

6 Conclusion and Future work

My framework consists of two main parts, feature extraction and malware prediction by machine learning technique. Three different ways of data pre-processing were used before training models. According to the final results, performances of all the models trained by dataset one is better than models trained by dataset two. I think this is due to the fact that some relevant information was dropped out during feature selection, therefore impaired model performances. Secondly, by comparing dataset 1 and dataset 3, it is obvious that the accuracies of GaussianNB and MLPClassifier models improved significantly. However, the accuracies of the other three models decreased. It may be due to that GaussianNB and MLPClassifier are more suitable for high-dimensional data, while the other three are more suitable for low-dimensional data. The reason behind this case will be a future work.

Besides the work I have done so far, other static features such as file bytecode features, strings, API call sequences can be used in the feature extraction part. Since there are some limitations of static features extraction, dynamic features extraction is a possible way to improve the performance. Integration of static and dynamic features extraction is promising and will another future work

References

[1] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in realtime," in *International workshop on recent advances in intrusion detection*. Springer, 2009, pp. 121–141.

[2] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," *None*, 2009.

[3] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 38–49.

[4] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 470–478.

[5] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[6] M. Belaoued and S. Mazouzi, "Statistical study of imported apis by pe type malware," in *2014 International Conference on Advanced Networking Distributed Systems and Applications*. IEEE, 2014, pp. 82–86.

[7] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.

```
dataset_encoder_train = dataset.drop(['Name','md5','Legitimate'],axis=1).copy()
train = dataset_encoder_train.sample(frac=.9, random_state=36)
test = dataset_encoder_train.loc[~dataset_encoder_train.index.isin(train.index)]

X_train = train.copy()
X_val = test.copy()

dfencoder_model = AutoEncoder(
encoder_layers = [32, 64, 128], #model architecture
decoder_layers = [], #decoder optional - you can create bottlenecks if you like
activation='relu',
swap_p=0.1, #noise parameter
lr = 0.01,
lr_decay=.99,
batch_size=128,
logger='ipynb', #special logging for jupyter notebooks
verbose=False,
optimizer='sgd',
scaler='gauss_rank', #gauss rank scaling forces your numeric features into standard normal distributions
min_cats=1, #Define cutoff for minority categories, default 10
progress_bar=False
)
dfencoder_model.device = 'cuda'
plt.figure(figsize=(24,10))
plt.rcParams['figure.figsize'] = [24, 12]
dfencoder_model.fit(X_train, epochs=80, val=X_val)
```

Figure 11. Example code of training a dfencoder model

[8] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian joint conference on artificial intelligence*. Springer, 2016, pp. 137–149.

[9] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE, 2015, pp. 11–20.

[10] T. Rezaei and A. Hamze, "An efficient approach for malware detection using pe header specifications," in *2020 6th International Conference on Web Research (ICWR)*. IEEE, 2020, pp. 234–239.