Comparative Analysis of Machine Learning Models with Different Types of Data Representations for Detecting Malicious Files

Alan Nafiiev [1], Hlib Kholodulkin [2], Andrii Rodionov [1] and Dmytro Lande [3]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Institute of Physics and Technology, Kyiv, Ukraine*
[2] *Taras Shevchenko National University of Kyiv, Faculty of Computer Science and Cybernetics, Kyiv, Ukraine*
[3] *Institute for Information Recording of National Academy of Sciences of Ukraine, Kyiv, Ukraine*

Abstract

Nowadays, malicious software authors are creating more and more advanced, sophisticated malware. To detect such programs, machine learning methods are increasingly used. However, these solutions can consume a lot of computing resources to perform their operations. Therefore, the problem arises of creating an optimal machine learning model regarding learning rate and the accuracy of malware detection. Also, usually one method is not enough for high-quality file detection, so it is more efficient to use several types of data representation. The purpose of this work is to conduct research on several machine learning models based on the support vector machine. Compare them with each other and identify the best model for two different types of data representation. The set of data used was collected from various Internet sources. It consists of 12824 executable files in .exe format, 11844 of which are malicious and 980 are benign. This article presents recommended methods for feature selection and input data generation for a machine learning model. These methods allow you to find the best option for preparing features that describe a malicious file, which will be used in the process of training and determining the model with the best parameters.

Keywords [1]

PE format, malware detection, feature selections, machine learning, intrusion detection.

CEUR-WS.org/Vol-3347/Short_8.pdf

1. Introduction

Recently, various methods for detecting malware that are not based on the signature approach have been increasingly proposed. These methods mainly use heuristic analysis, behavioral analysis, or a combination of both to detect malicious software [1]. Such methods are being actively researched due to their ability to detect zero-day malware without any prior knowledge of it [2, 3]. These approaches often involve using different machine learning models. However, their use always entails several important challenges that require careful study. Firstly, there is the task of determining and extracting a suitable set of features that will display the input data set in the best way. Secondly, these features must be correctly formed for the machine learning model so that the operations use as little time and computing resources as possible. This work is aimed at solving these two tasks. In recent years, many studies have been published offering various solutions to these issues [4, 5, 6]. In these works, fairly common methods of feature extraction were proposed. In our last article, we analyzed a set of files using various machine learning algorithms, where the binary code of the file was used as the features of the machine learning model [7]. To obtain features, we have presented the executable file in PE format. Also, we have selected the most optimal parameters for each of the classification algorithms. The support vector algorithm showed good accuracy results and showed the promise of its use in further research. More specifically, the research described in this article focuses on the following points:

Information Technology and Implementation (IT&I-2022), November 30 - December 02, 2022, Kyiv, Ukraine
EMAIL: Alan.nafiev@gmail.com (A. Nafiiev); kholodulkin.hlib@gmail.com (H. Kholodulkin); dwlande@gmail.com (D. Lande); andrey.rodionov@gmail.com (A. Rodionov)
ORCID: 0000-0003-3945-1178 (D. Lande)

© 2022 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

1. Determine the approach for generating input data for a machine learning model.
2. Test the resulting models and use several metrics to identify the optimal model for detecting malicious files.

The structure of the article is implemented as follows: Subsection 1.1 introduces general information about the PE format.

Section 2 describes the research dataset, how it was generated, and what types of files it contains. Different types of data representation and feature extraction methods are described in Section 3. Section 4 describes the support vector machine. The conducted experiments and comparison of the results of the accuracy of machine learning models are described in Section 5. The evaluation of the results of the analysis of experimental data is described in Section 6. Section 7 concludes this article, which contains the conclusions of the studies.

2. Portable Executable Format

Most malware targets the Windows operating system. The Portable Executable (PE) file format is a standard format used by executable files and dynamic link libraries in the Windows operating system. In addition to the binary code itself, the PE file contains structural information, describes how the OS should map the program in memory, and which external functions are called through the import address table [8]. Features can be extracted from this metadata in the form of *n*-grams of bytes or *n*-grams of instructions [9]. Examining these features indicates the behavior of the program at run time and allows you to distinguish between malicious and harmless Windows binaries. Figure 1 shows a diagram of a PE file.

3. Dataset Construction

The most important step in building a machine learning model is to create an input dataset. It is also necessary that such a set be as well balanced as possible in order to obtain the most representative results. The input data was presented by a data set containing 30750 .exe files that were collected from various Internet sources. Next, you need to filter the data set by removing identical files from it. For each file, a sequence of its bytes was collected. Subsection 3.1 details this process. The identity of the sequence is determined based on all the bit information that could be collected from the file: DOS_HEADER, FILE_HEADER, OPTIONAL_HEADER, SECTION_HEADER, Export, Import and also .rsrc section. Basically, in a set of viruses there are duplicates under different names and very similar files.

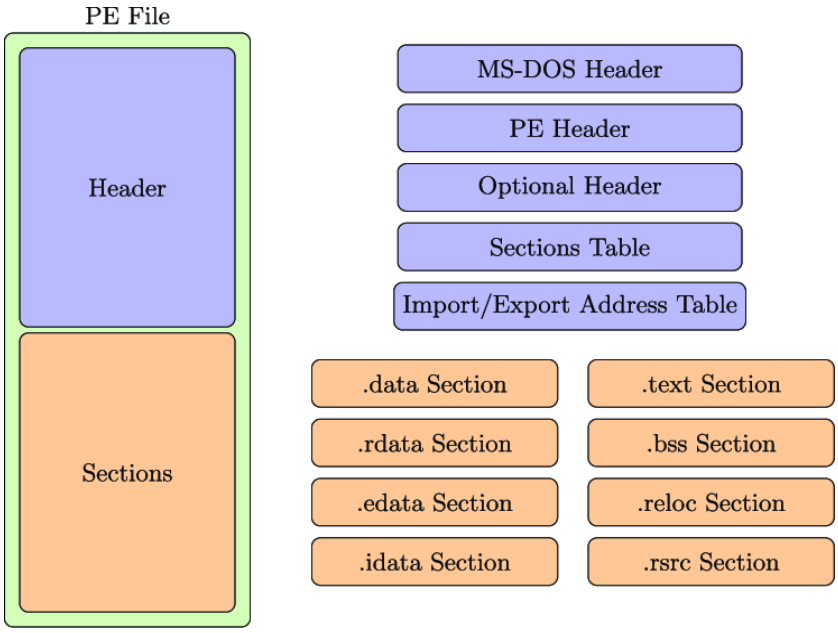

Figure 1: PE file diagram

After files with identical sequences were removed from the set, the data set used in this study was obtained, which contains 11844 malicious and 980 benign .exe files. Malware files are divided into 5 types: Locker (300), Mediyes (463), Winwebsec (657), Zbot (1973), Zeroaccess (407), CryptoRansom (5856), Zeus (1413), InstallCore (731). The distribution infographic can be seen in Figure 2. It is worth noting that in the data set used, there are much fewer benign files than malicious ones. On a real system, this can be a problem, but in our case, such a distribution is acceptable, since our main task is to compare the methods of generating features with each other. All malware files were taken from the websites «virusshare.com» and «malicia-project.com». The benign files were taken from the installed application folders of legal software from different categories. They can be downloaded at «download.cnet.com/windows».

4. Feature Selection

As discussed earlier, PE executable files contain a lot of different kinds of information that can be used to detect malware. We can view each file as a series of bytes, a trace of instructions or function calls. Different file representation types require different approaches to feature extraction, selection, and preprocessing.

4.1. Binary

To generate input data for a machine learning model, you need to extract useful bit information from the executable file. To do this, each file is processed in turn, using the *pefile* library from Python. The functions of this module allow you to get the bytes of specific fields from various sections of the PE file as a string. For each file from the dataset, a list is formed containing a sequential sequence of bytes of the file. A visualization of this process can be seen in Figure 4. In this study, two models of binary representation of data are considered. One model only uses the byte sequence from the Header: DOS_HEADER, FILE_HEADER, OPTIONAL_HEADER, SECTION_HEADER, Export and Import.

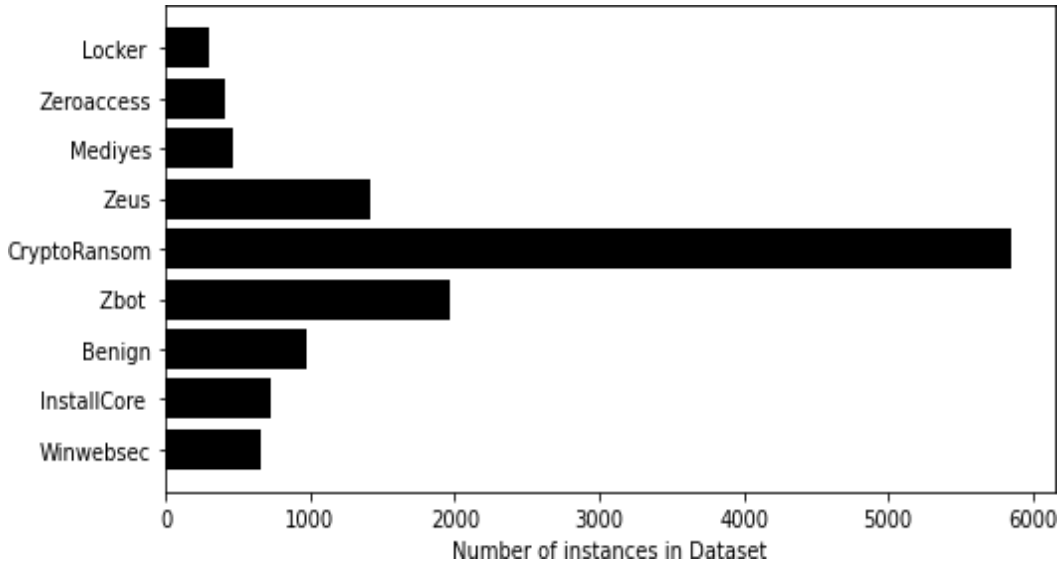

Figure 2: Distribution of files in the dataset

The second model uses all the Header and .rsrc section since it is available in the pefile module. Below you can see an example of the fields used for the DOS Header [10]:

```
typedef struct _IMAGE_DOS_HEADER {
    WORD e_magic;        /* 00: MZ  Header signature */
    WORD e_cblp;   /* 02: Bytes on last page of file */
    WORD e_cp;     /* 04: Pages in file */
    WORD e_crlc;   /*  06:  Relocations  */
    WORD e_cparhdr;      /* 08: Size  of  header  in  paragraphs  */
    WORD e_minalloc;     /* 0a: Minimum extra paragraphs needed */
```

```
    WORD e_maxalloc;    /* 0c: Maximum extra paragraphs needed */
    WORD e_ss;      /* 0e: Initial (relative) SS value */
    WORD e_sp;      /* 10: Initial SP value */
    WORD e_csum; /* 12: Checksum */
    WORD e_ip;      /* 14: Initial IP value */
    WORD e_cs;      /* 16: Initial (relative) CS value */
    WORD e_lfarlc;      /* 18: File address of relocation table */
    WORD e_ovno; /* 1a: Overlay number */
    WORD e_res[4];      /* 1c: Reserved words */
    WORD e_oemid;       /* 24: OEM  identifier (for e_oeminfo) */
    WORD e_oeminfo;     /* 26: OEM information; e_oemid specific */
    WORD e_res2[10];    /* 28: Reserved words */
    DWORD e_lfanew;    /* 3c: Offset to extended header */
} IMAGE_DOS_HEADER,  *PIMAGE_DOS_HEADER;
```

$N-$gram representation was one of the first feature generation methods used for malware detection [11]. An important difference that this study contains is the use of transition probabilities to build a Markov chain instead of 2-grams as the underlying representation of the data. Our Markov chain can be represented as a graph in which the vertices are the states of the process (all possible 256 bytes), and the edges are transitions between states, and p_{ij} is written on the edge from i to j – the probability of transition from one byte to another.

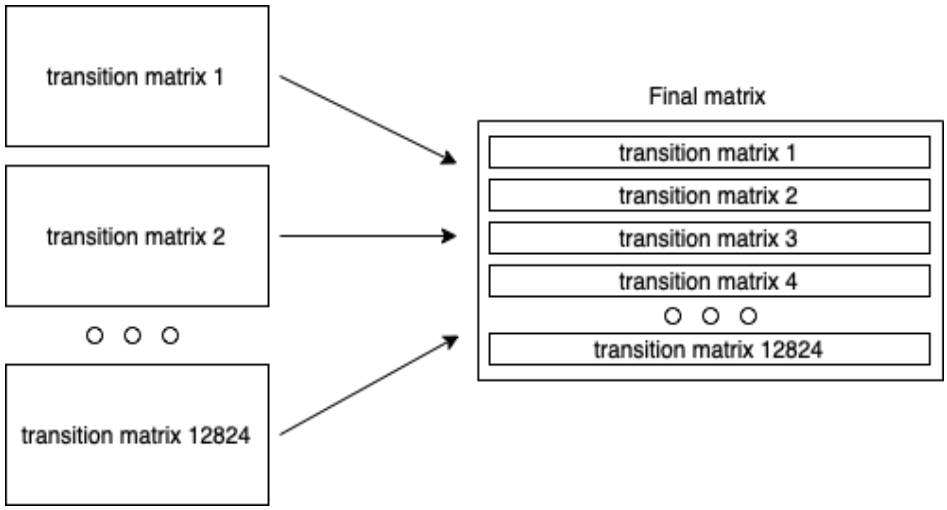

Figure 3: Final matrix formation diagram

As a result, we get a quadratic transition matrix $P = p_{ij}$, dimension 256*256, on which the following conditions are imposed:

$$p_{ij} \geq 0, . \quad (1)$$

$$\forall i \sum_{j} p_{ij} = 1 . \quad (2)$$

To construct such a matrix of transition probabilities for each byte to each, first, based on the collected sequence of bytes for all files, it is necessary to construct an adjacency matrix of the form (00, 01, ..., ff) * (00, 01, ..., ff), where for each pair of bytes the matrix counts how many times the first byte was immediately followed by the second byte. Then, after the formation of adjacent matrices, transition matrices are constructed for all files.

After that, the final matrix of dimension 12824 (the number of all files)*256^2 is formed, in which each line corresponds to one file and includes its transition matrix translated into vector form. Each such vector contains 65536 values. A diagram of this process can be seen in Fig 3. Before passing the final matrix to the machine learning algorithm, you first need to divide this matrix into a training set

and test set. Since our dataset does not include many different types of malicious files, it was decided to use 40 % of the final matrix for training and 60 % for the test.

4.2. Trace

To detect malware, executable file instruction tracing is also used. The disassembled code is generated by the IDA Pro program, which is opened via the command line using a python script. The program saves the disassembled instructions of each element into .asm files. An example ASM file is shown in Figure 5.

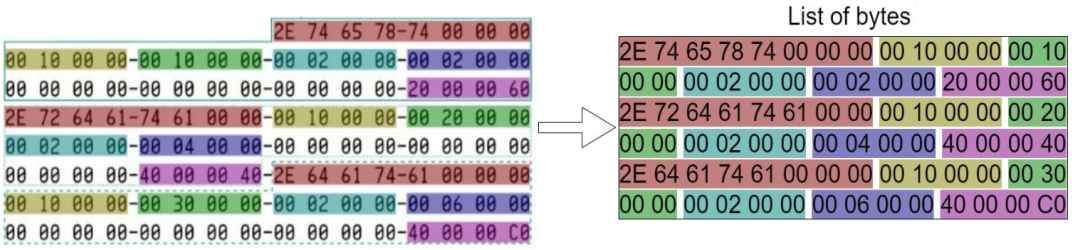

Figure 4: The process of collecting features for a model with a binary representation of data. The pefile package functions parse bytes, which are sequentially assembled into a list

```
; ---------------------------------------------------------------------------

loc_4112BC:                             ; CODE XREF: sub_411239+70\u2191j
                mov     [ebp+var_48], 0
                mov     [ebp+var_48], 1
                push    14h             ; dwMilliseconds
                call    ds:Sleep
                call    sub_411565
                mov     [ebp+var_48], eax
                cmp     [ebp+var_48], 0
                jz      short loc_4112FB
                mov     eax, [ebp+var_28]
                mov     [ebp+var_24], eax
                mov     [ebp+var_4], 3D63h
                mov     ecx, [ebp+var_24]
                push    ecx
                call    sub_411000
                add     esp, 4
                jmp     short loc_411302
; ---------------------------------------------------------------------------
```

Figure 5: Fragment of an ASM file. Instructions used are outlined in red

Table 1
Number of features

N	Number of features
50	272
400	187
2500	126
9500	74

After that, for each file, instructions are parsed in turn (mov, push, call, jz, etc.), which are collected in one large array. From this array, instructions are removed that occur in total in all files less than a certain threshold (N = 50, 400, 2500, 9500). Table 1 shows the ratio of the parameter N to the dimension

of the adjacency matrix. As Figure 6 shows, increasing this hyperparameter can significantly reduce the dimension of the final matrix and, consequently, reduce the training time of the machine learning model.

After that, an adjacency matrix is built for each file in the same way as in the binary representation. Unique instructions define the dimension of a two-dimensional quadratic adjacency matrix, in which for each pair of instructions in the matrix, it is counted how many times the second instruction immediately followed the first one. Finally, transition matrices are constructed using adjacency matrices.

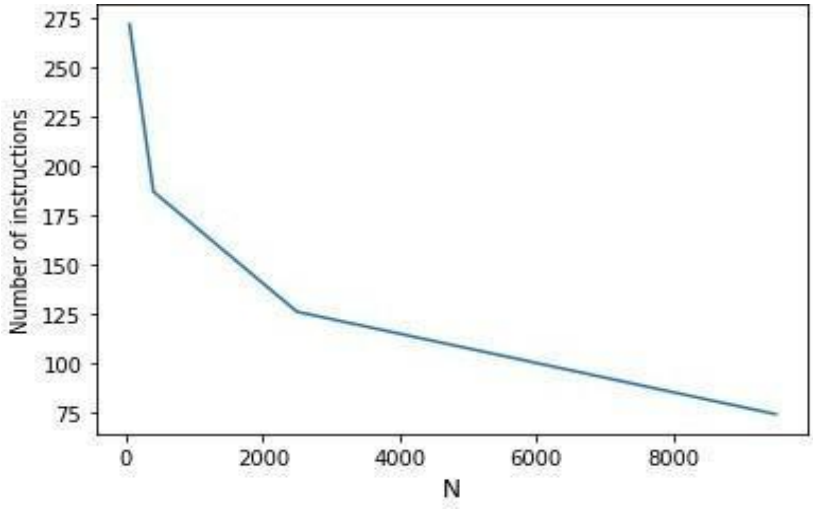

Figure 6: The dependence of the hyperparameter N on the number of instructions used

The transition matrices are added into vectors, forming the final matrix, as in Fig 3. The number of features depends on the parameter N, which indicates the number of deleted instructions. As in the binary representation of the data, the final matrix is divided into two sets – training set and test set, with a ratio of 60 % of the test set.

5. Support vector machine

The support vector machine has been chosen as one of the most popular algorithms. According to numerous studies [12, 13], it shows good results specifically for binary classification. In the classical SVM based on kernels, the weight vector is trained, which determines the contribu- tion of each sample to the separating hyperplane between classes, by solving the following optimization problem:

$$\left(\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K\left(x_i, x_j\right) - \sum_{i=1}^{n} \alpha_i \right) = S_k\left(\alpha\right). \tag{3}$$

With restrictions:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, . \tag{4}$$

$$0 \le \alpha_i \le C . \tag{5}$$

The model is trained using the SVM algorithm with a square exponential kernel:

$$K\left(x, y\right) = \exp\left(-\lambda \left\| x - y \right\|^2\right). \tag{6}$$

The algorithm receives 40 % of the final matrix as input, where each row contains the transition matrix of one file. Hyperparameters are selected using cross-validation.

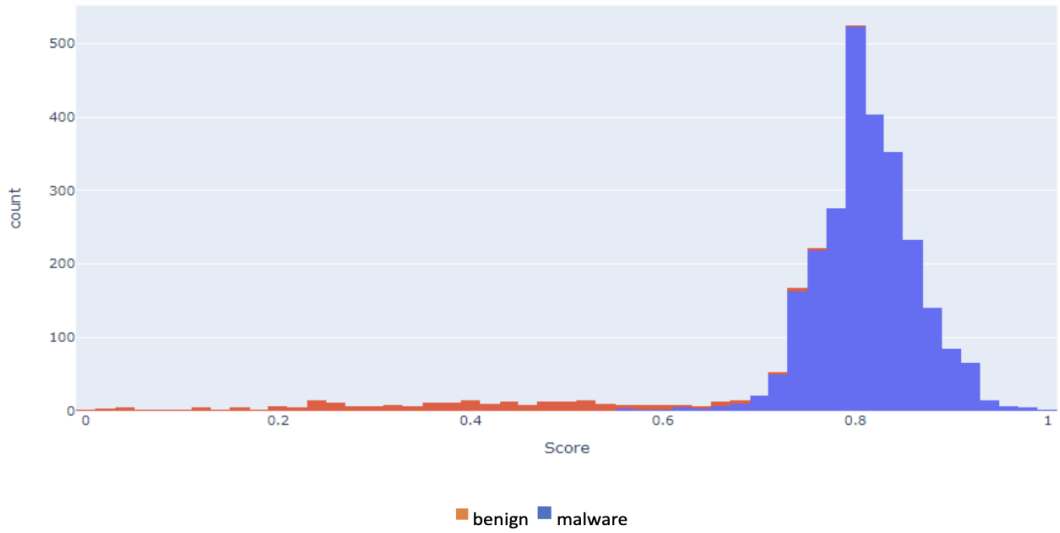

Figure 7: Histogram of file label probability estimates for the Trace_272 model

6. Results

Tests were carried out for two types of data presentation with a ratio of 40 % of the test to training set. The indicators of the main metrics can be observed in Table 2.

The results show that the number of features used has little effect on the result of the classifier. However, it is still possible to isolate the best model by relying on the F-score metric, which is a harmonic average between recall and precision. The highest result for the tracing type of feature collection was shown by the variant with 74 instructions (Trace_74). On Fig 8 you can observe the ROC and Precision-Recall curves. Among the binary type of data representation, a model that includes a resource section (Binary_r) showed a slight advantage in all metrics. However, the training time also increased as more data was used. In addition, it is worth noting that the binary representation of the data showed a comparable accuracy result with the best model trained on instruction traces.

7. Discussion

In this study, we created 6 machine learning models, four of which are trace type feature collection, and two are binary. Based on the results, the optimal model for the tracing type of data representation will be a model with 74 instructions. In addition, the training time of this model is much less than that of the variants with 126, 187 and 272 features, which is a significant plus. Therefore, it can be argued that the principle of using all possible data to train the model does not always give the best accuracy result. Sometimes it is useful to select a small part of the entire set of features on which the trained model will show the highest accuracy result.

Table 2

Binary_r – a model that includes the data section of the file. Model Binary_wr does not. Time is measured in seconds

	recall		precision		F-score		roc_auc	pr_auc	Time
	0	1	0	1	0	1			
Binary_wr	0.9130	0.9909	0.8873	0.9932	0.9000	0.9921	0.9956	0.9996	795
Binary_r	0.9239	0.9915	0.8947	0.9940	**0.9090**	**0.9928**	**0.9961**	0.9997	816
Trace_272	0.8369	0.9882	0.8461	0.9873	0.8415	0.9877	0.9908	0.9993	747
Trace_187	0.8695	0.9887	0.8571	0.9898	0.8633	0.9893	0.9906	0.9992	490
Trace_126	0.9094	0.9873	0.8479	0.9929	0.8776	0.9901	0.9921	0.9994	255
Trace_74	0.9130	0.9893	0.8689	0.9932	**0.8904**	**0.9912**	**0.9946**	0.9995	99

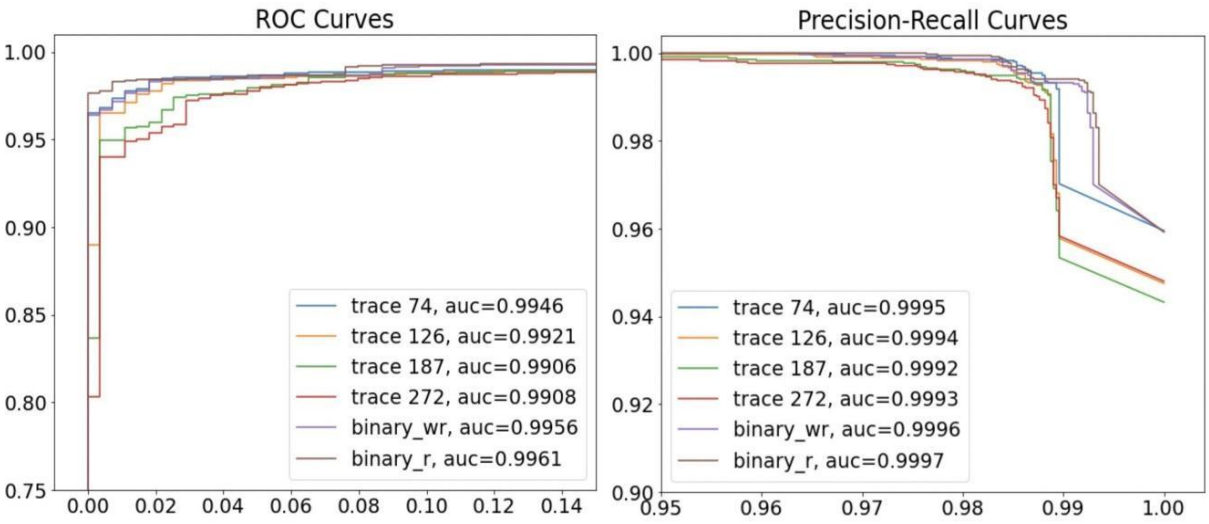

Figure 8: ROC and Precision-Recall curves

In the binary form of the data representation, the model, which includes the file data section, showed a slightly higher result for all metrics. However, the training time increased slightly. Therefore, for this type of data representation, the model with a data section will be optimal for use in real malware detection systems.

8. Conclusions

In this work, we have created machine learning models derived from two different approaches for feature extraction. The first method was the bit information of a file with a PE structure. Using the pefile library, bytes were parsed from various sections of the file, which were then used in the construction of transition matrices used to train the model. The second approach used executable file instruction traces. The disassembled code was generated by IDA Pro. Then instructions were parsed and, as in the binary representation of data, adjacency matrices were built for each file using 2-grams. All models demonstrate a very good result in detecting malicious files.

Figure 9: Binary_r model confusion matrix

The best accuracy indicators were demonstrated by models trained on a binary representation of data. The model containing the resource section scored the highest across all metrics. In particular, the metrics F0-score – 0.9090, F1-score – 0.9928 and roc_auc – 0.9961. The results of the accuracy of our models may seem too high, but it is worth considering some limitations. Firstly, the dataset used is not

well balanced. There are an order of magnitude fewer benign files than viral ones, so the machine learning model will be better at recognizing malicious files. In the framework of our study, this is acceptable, since our goal is to compare all models with each other and identify the best one. However, on a real file recognition system, a balanced and carefully filtered data set should be used. Secondly, there are not many different types of malicious files in the collected dataset. Viruses of the same type are very similar to each other, which leads to the fact that it is easy for the machine learning algorithm to recognize such files. Future studies should use as many varieties of malicious files as possible to obtain a more representative result of detection accuracy.

We also managed to identify the optimal variant of the number of trace instructions used for training the model using the support vector machine. The best was the model trained on the least number of instructions – 74 (roc_auc – 0.9946). In addition, this model requires the least time for its training, which is a significant plus for using the model in real file classification systems.

It is worth noting that in this paper we have chosen a support vector machine algorithm for building machine learning models. One of the main reasons for this choice is future research. In the dissertation we use 6 types of transformations of exe files, both static and dynamic. And in order to combine all the branches of file transformations into one generalized machine learning model, the issue of data matching arises. To solve this task, we decided to use SVM, since this algorithm uses a kernel, which can be used to combine different types of data.

9. References

[1] F. Abri, S. Siami-Namini, M. A. Khanghah, F. M. Soltani, A. S. Namin, The performance of machine and deep learning classifiers in detecting zero-day vulnerabilities (2019).

[2] S. K. S. Anand Hand, Ashu Sharma, Machine learning in cybersecurity: A review, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (July 2019).

[3] S. Sharma, C. R. Krishna, Zero-day malware detection based on supervised learning algorithms of api call signatures, Australasian Data Mining Conference (AusDM11), December 2011.P. Chaudhary, Pe file-based malware detection using machine learning (January 2021).

[4] P. Chaudhary, Pe file-based malware detection using machine learning (January 2021)

[5] A. Kutlay, K. Karađuzović-Hadžiabdić, Static based classification of malicious software using machine learning methods (2020).

[6] O. N. U. Hasan H. Al-Khshali, Muhammad Ilyas, Effect of pe file header features on accuracy, 2020 IEEE Symposium Series on Computational Intelligence (SSCI), December 2020.

[7] A. R. Alan Nafiiev, Hlib Kholodulkin, Comparative analysis of machine learning methods for detecting malicious files, Theoretical and Applied Cybersecurity, Algorithms and methods of cyber attacks prevention and counteraction (2022).

[8] F. M. M. F. M. Zubair Shafiq, S. Momina Tabish, Pe-miner: Mining structural information to detect malicious executables in realtime (September 2009).

[9] S. L. Imad Abdessadki, A new classification based model for malicious pe files detection, International Journal of Computer Network and Information Security (June 2019).

[10] S. Dösinger, 2016. https://github.com/wine-mirror/wine/blob/master/include/winnt.h# L2253.

[11] R. C. Edward Raff, Richard Zak, other, An investigation of byte n-gram features for malware classification, Journal of Computer Virology and Hacking Techniques (February 2018).

[12] Y. Lifshits, Algorithms for internet: Support vector machines (2006).

[13] Caruana, An empirical comparison of supervised learning algorithms (2006).