DOI: https://doi.org/10.15276/aait.05.2022.25
UDC 004.89

Semantic analysis and classification of malware for UNIX-like operating systems with the use of machine learning methods

Maksym V. Mishchenko[1)]
ORCID: https://orcid.org/0000-0001-9769-9759; it144111@stu.cn.ua
Mariia S. Dorosh[1)]
ORCID: https://orcid.org/0000-0001-6537-8957; mariyaya5536@gmail.com. Scopus Author ID: 56912183600
[1)] Chernihiv Polytechnic National University, 95, Shevchenko Str. Chernihiv, 14035, Ukraine

ABSTRACT

The paper focuses on malware classification, based on semantic analysis of disassembled binaries sections' opcodes with the use of n-grams, TF-IDF indicator and machine learning algorithms. The purpose of the research is to improve and extend the variety of methods for identifying malware developed for UNIX-like operating systems. The task of the research is to create an algorithm, which can identify the types of threats in malicious binary files using n-grams, TF-IDF indicator and machine learning algorithms. Malware classification process can be based either on static or dynamic signatures. Static signatures can be represented as byte-code sequences, binary-assembled instructions, or imported libraries. Dynamic signatures can be represented as the sequence of actions made by malware. We will use a static signatures strategy for semantic analysis and classification of malware. In this paper, we will work with binary ELF files, which is the most common executable file type for UNIX-like operating systems. For the purpose of this research we gathered 2999 malware ELF files, using data from VirusShare and VirusTotal sites, and 959 non malware program files from /usr/bin directory in Linux operating system. Each malware file represents one of 3 malware families: Gafgyt, Mirai, and Lightaidra, which are popular and harmful threats to UNIX systems. Each ELF file in dataset was labelled according to its type. The proposed classification algorithm consists of several preparation steps: disassembly of every ELF binary file from the dataset and semantically processing and vectorizing assembly instructions in each file section. For the setting classification threshold, the Multinomial Naive Bayes model is used. Using the classification threshold, we define the size for n-grams and the section of the file, which will give the best classification results. For obtaining the best score, multiple machine learning models, along with hyperparameter optimization, will be used. As a metric of the accuracy of the designed algorithm, mean accuracy and weighted F1 score are used. Stochastic gradient descent for SVM model was selected as the best performing ML model, based on the obtained experimental results. Developed algorithm was experimentally proved to be effective for classifying malware for UNIX operating systems. Results were analyzed and used for making conclusions and suggestions for future work.

Keywords: Malware detection; machine learning; semantic analysis; multiclass classification; text mining; operating system

For citation: **Mishchenko M. V., Dorosh M. S. "Semantic analysis and classification of malware for UNIX-like operating systems with the use of machine learning methods".** ***Applied Aspects of Information Technology.*** **2022; Vol.5 No.4: 371–386. DOI: https://doi.org/10.15276/aait.05.2022.25**

INTRODUCTION

Modern malware is evolving at a very fast pace, and new cyber threats appear every day. For more effective detection of malicious software, antivirus programs widely use artificial intelligence methods – both separately and in combination with other cyber security approaches. Among the popular approaches is the adaptation of natural language processing (NLP) techniques to binary files or their metadata.

The resulting vectorized texts are used for classification using machine learning methods. For example, in 2021, a new malware detection algorithm using NLP and machine learning was developed for the Microsoft 365 Defender antivirus [1]. Its essence is to apply the word embedding technique to fuzzy hash sums of viruses for further classification using a multilayer perceptron. Thanks to this algorithm, the antivirus managed to identify a new variation of the GoldMax virus, which was later confirmed and published.

The use of convolutional neural networks (CNN) for the classification of malicious software is gaining popularity. Usually, CNNs are used for pattern recognition and working with images, but their architecture allows processing raw bytes of any binary files, which gave the stimulus for their adaptation in the field of identification and classification of malicious software [2]. For implementation in antivirus software, models are usually used that analyze the file as a whole, without analyzing the structure of each file. This makes it possible to reduce resources for the development and training of algorithms. At the same time, in order to achieve high accuracy of such algorithms, it is necessary to generate a large amount of input data, using a large number of malicious program files as samples. With access to antivirus software databases, collecting a large dataset of malicious software files is not a difficult task, but in other cases it is one of the most difficult and long steps of preparing for machine learning. To reduce the amount of input data and avoid loss of machine learning accuracy, it is necessary to increase the number of significant features of the input data. This can be achieved by more detailed analysis and selection of the most important parts of the program file.

Thus, the motivation for carrying out this study was the possibility of identifying malicious software on relatively small datasets through more detailed processing of each of the files. It was decided to conduct a study of the content and purpose of each section of the program file and highlight the most significant for classification using machine learning. Also, instead of raw bytes, it was decided to use assembly commands of the selected program section, performing semantic analysis, text vectorization and further classification by machine learning methods.

LITERATURE ANALYSIS

Malware detection and investigation can be broadly divided into two approaches: static analysis and dynamic analysis. [3] Static analysis includes methods of examining bytecode, assembly binary commands, or imported dynamic libraries (DLLs). Dynamic analysis involves studying the behavior of malicious software. Static analysis can be applied both to the entire binary file and to the file's sections.

For example, Ian Shiel et al. in their work [4] proposed a method of improving the fuzzy hashing algorithm by applying it to each section of a binary PE file. With their research, the authors solve the problem of detecting malicious software for the Windows OS, in which, by design, there are sections common to all files, developers can change the order of program sections or insert additional sections to complicate its identification. As a result, the authors achieved 92% more true positive (TP) detections on unobfuscated files and 88% more TP for packaged malware, compared to fuzzy hashing of the whole file.

The application of Convolutional Neural Networks (CNN) to binary files for their classification is well researched. For example, Edward Raff et.al.
in their study [5] used an entire binary file in the form of a sequence of bytes, which was fed to the input of a convolutional neural network for further classification. One of the problems with working with a raw byte stream has been the fact that bytes have different meanings depending on the context. By applying a word embedding algorithm to the byte stream, which allowed them to highlight byte words with similar meaning and context, they managed to solve that problem. The developed MalConv architecture [5] showed good generalized results on large data datasets and can be applied to binary files without focusing on their internal structure. A significant drawback of the proposed solution is the large time and computing power required to perform convolution operations on very long data, such as the output byte stream of a binary file.

Another example of static analysis of binary files using CNN is the work of Fangtian Zhong et al. [6], in which the authors proposed to transform the input binary file into an image that is processed by a contrast-limited adaptive histogram equalization algorithm and classified by CNN. The resulting model showed good results and efficiency.

The main advantage of approaches with convolutional neural networks is the absence of the need for domain knowledge of cyber security and detailed study of the file structure. At the same time, such approaches do not take into account the peculiarities of building binary files and have high resource costs for training and calibrating machine learning models.

Another approach to malware analysis is to apply word embedding techniques and semantic analysis to the text or byte content of a file.

For example, BooJoong Kang et al. in their study [7] applied the n-gram method to analyze and process the operation codes of APK files. The resulting n-grams were used for machine learning models. The authors achieved the best classification performance using n=3 and n=4 n-grams.

Semantic analysis is also used in combination with dynamic analysis of malicious software, studying and analyzing the results of execution of malicious files. For example, in their study [8], Bin Qin et al. applied the TF-IDF method to the sequence of API calls in application log files and used this information as a dataset to identify and classify malicious and safe software. The results of the study proved the effectiveness of the application of semantic analysis and the TF-IDF method to improve the results of classification and detection of malicious software.

FORMULATION OF THE PROBLEM

The problem of classification of malicious software is actively researched. PE files, which are executable files for the Windows operating system, or APK files, which are executable files for the Android operating system, are usually taken as the research object. The reason for this, among others, is the large volume of virus types developed for these platforms. ELF files, which are executable files for UNIX-like operating systems, are less often used as objects for malware software research and classification. However, there are currently quite threatening and destructive types of malware for UNIX-like operating systems, including the most common Ransomware, Worm, Trojan and BotNet. The relatively small number of works [9] [22] investigating threats to UNIX-like operating systems is a problem, and this work contributes to solving this problem.

THE PURPOSE AND THE OBJECTIVES OF THE STUDY

Malware for UNIX-like operating systems was chosen as the object of this study.

The purpose of this research is to expand the methods of identifying malicious software for UNIX-like operating systems using semantic analysis and classification by machine learning models.

To achieve the goal of the research, several tasks were set. First, collecting malicious and safe binaries for UNIX-like operating systems and labeling each file accordingly. Second, vectorization of assembly commands of disassembled binary files and classification of these program files according to their label. Third, determination of feasibility of application and comparison of accuracy and time spent of various machine learning methods for classification of malicious software.

RESEARCH METHODS

For the purpose of the research, we needed to select appropriate semantic analysis methods and machine learning models for the classification. These problems were solved by analyzing relevant software libraries and literature [11,12], [13,14], [15,16], [17,18], [24]. As a result, Multinomial Naïve Bayes, Support vector machine, stochastic gradient descent and gradient boosting models were selected for running the classification.

The n-gram method and the TF-IDF algorithm were considered as methodologies for vectorization of text data.

The problematic aspect of the research was collecting and obtaining the information about binary files for identifying malicious software. This is due to the limited number of datasets available where the raw binary ELF file is flagged as a specific type of malware. A set of raw binary ELF files was downloaded from the VirusShare site [19], and labels with types for each file were placed using the public API of the VirustTotal site [20].

The manual about the portable file formats [10] helped to investigate the structure of executable ELF files and their sections.

CHOOSING SEMANTIC ANALYSIS METHODS

The first stage of the research is the semantic analysis of the program file section. As input data for semantic analysis, it was decided to take a sequence of assembly commands located in different sections of the binary file. At the output, it is necessary to obtain numerical vectors for further classification.

One of the basic approaches to natural language processing (NLP) is the n-gram method [11]. This method is aimed for creating a probabilistic model for predicting the next phrase based on statistical indicators. N-gram models predict word w_i based on the sequence $\{w_{i-(n-1)},...,w_{n-1}\}$. For computing the probability of the sequence $P(w_{1,}w_2,...,w_n)$ formula 1 is used.

$$P(w_{1:n}) = \prod_{k=1}^{n} P(w_k \mid w_{1:k-1}), \quad (1)$$

where $P(w_k \mid w_{1:k-1})$ is conditional probability of appearing of the word w_k in the sequence of the words $w_{1:k}$.

The n-gram method is an important step in the processing of program code because it groups together semantically important parts, such as an assembly instruction and its operands.

Application of the n-gram method does not take into account the length of the document. In longer documents, n-grams may occur more often than in shorter ones, while both documents may have a common content. Therefore, in our research, n-gram will be used as the basis for the TF-IDF method, which takes into account the length of documents and the frequency of words in them. TF-IDF is a statistical indicator that allows to determine which word is used most often in a specific document and less often in all other documents of the collection [14]. The indicator consists of two parts: TF (term frequency) – word frequency calculated by relation 2.

$$TF = \frac{n_i}{\sum_{k=1}^{N} n_k}, \quad (2)$$

where n_i is amount of word’s appearing in the docu-ment; n_k is amount of the k-word appearing in the document; N is total amount of the words in the document.

Inverse document frequency (IDF) – the inverse value of the frequency with which the word occurs in all documents of the collection is calculated by relation 3.

$$IDF = \log \frac{|D|}{|d_i \supset t_i|}, \quad (3)$$

where $|D|$ – amount of the documents in the collection; $|d_i \supset t_i|$ – amount of the documents d_i, which have word t_i.

As a result of the product of two indicators, we will get the TF-IDF value (4).

$$\text{TF-IDF} = \text{TF} \cdot \text{IDF}, \quad (4)$$

where TF is term frequency; IDF is inverted document frequency.

Therefore, the TF-IDF value is directly proportional to the number of uses of the selected word in the selected document, and inversely proportional to the number of documents containing the selected word.

In our work, we investigated the size of n-grams from 1 to 5 and chose the one that gives the highest accuracy for the basic classification threshold. The generated n-grams for the selected section of the file were used for further vectorization by the TF-IDF method.

CHOOSING MACHINE LEARNING MODELS

Polynomial Bayes classifier (Multinomial Naive Bayes) was used to establish the basic classification threshold. The main idea of this method is to find the class to which the document belongs with the highest probability, which is calculated according to the Bayesian formula (5) [16].

$$P(c \mid t_i) = \frac{P(c)P(t_i \mid c)}{P(t_i)}, \; c \in C, \quad (5)$$

where C – set of documents classes; t_i – document of the collection.

Having proposed the hypothesis that the words in the documents are distributed using a certain parametric model, we can determine these parameters using the Polynomial Bayesian classifier. For example, based on this statement, Jiang Su et al. solved the problem of text classification in their work [28].

As the main classification methods, support vector models, the gradient descent method and the gradient boosting method were chosen.

Models based on the method of support vectors - support vector machine or SVM, are often used for text classification, because they are optimized for detecting non-linear patterns in the multidimensional space of features, which is a vectorized text [12]. The accuracy of SVM classification depends on the selected kernel function. For comparison, 4 functions were investigated: linear, polynomial, radial and sigmoid.

Two methods were chosen to improve training results: gradient descent and gradient boosting. The purpose of the gradient descent method is to find the local minimum of the differentiated function [13]. This method is used to minimize the loss function during model training by step-by-step reconfiguration of model parameters.

The method of gradient boosting consists in using the composition of models [17, 18]. In the process of finding the local minimum of the loss function, the best model from the composition is selected.

Stochastic gradient descent (SGD) is aimed at minimizing the loss function during model training. Unlike regular gradient descent, stochastic gradient descent uses randomly selected instances of the training sample at each iteration, instead of finding the minimum
for all sample instances, which optimizes its performance [15]. In our work SVM loss function will be the objective for SGD algorithm.

Algorithm XGBoost [18], which uses a composition of decision tree models, will be used as a gradient boosting method.

In the course of the study, an algorithm was created that classifies malicious and safe software using a prepared set of malicious and safe software files. The result of the algorithm is a set of machine learning models trained to recognize malicious and safe software for UNIX-like operating systems. The scheme of the created algorithm is shown in Fig. 1.

CREATING A DATASET

As input data for the algorithm, it was decided to use a set of binary ELF files that were identified as malicious and safe.

An ELF file (Extensible Linking Format) is a format for binary executables, object modules, or libraries for UNIX-like operating systems. Each ELF file consists of two parts: ELF header and file data. The ELF header section defines the format of a particular file and is always zero-indented. The file data section can contain a table of program headers and a table of program sections [10].

In this work, the part of file data with the sections of the program will be investigated. Each of the sections has a different purpose and content. The main sections of ELF files and their purpose are listed below:

– .text – contains program code;

– .data – contains initialized data of the program;

– .rodata – contains initialized read-only data;

–.bss – contains not initialized variables of program.

Each of the listed sections of the program file was semantically analyzed. For use in the developed algorithm, the section that showed the best classification results in combination with the appropriate number of n for n-gram was selected when determining the classification threshold.Binary ELF files were downloaded from the VirusShare site [19]. For each of the files, information was obtained about the existing threats that the file carries. For this, the free public API of the VirusTotal site was used [20]. Using the MD5 hash sum of each file, a report on the file and its threats detected by various anti-virus software was processed. After examining the number of detected threats for each of the antivirus software, the Microsoft antivirus was chosen as the one with the most detected threats that

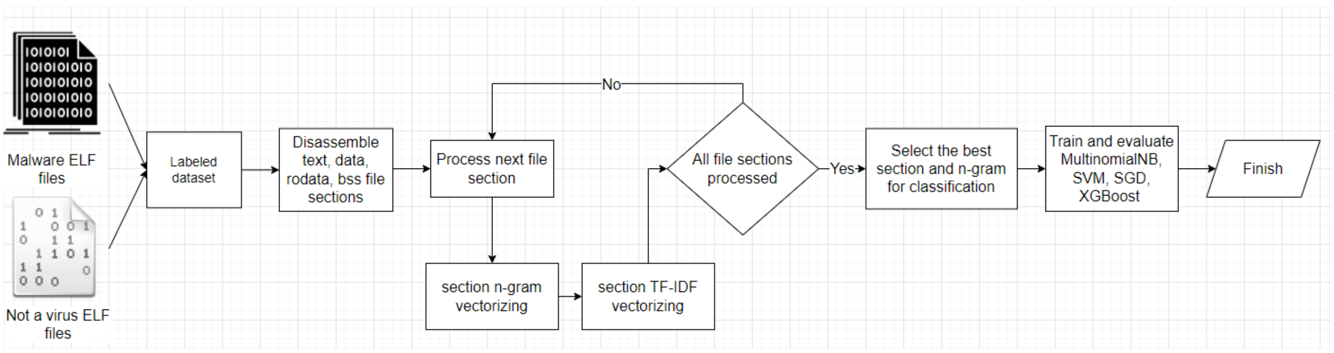

Malware ELF files
Not a virus ELF files
Labeled dataset
Disassemble text, data, rodata, bss file sections
Process next file section
No
All file sections processed
Yes
Select the best section and n-gram for classification
Train and evaluate MultinomialNB, SVM, SGD, XGBoost
Finish
section n-gram vectorizing
section TF-IDF vectorizing

Fig.1. **Schema for the algorithm for semantic analysis and classification of malware for UNIX-like operating systems using machine learning methods**
***Source*: compiled by the authors**

can be grouped into separate families. Threats that are the most common and have enough examples to form a dataset were chosen for the study. Different threat mods that belong to the same family have been grouped together and labeled with their family name. In this way, it was possible to obtain a balanced dataset with three main families of threats.

– Gafgyt – malware, that infects UNIX-like operating systems and runs DDoS attacks from them [22].

– Mirai – virus, that infects UNIX-like operating systems and create bot-net from them. Usually running on the internet of things [21].

– Lightaidra – virus, that infects UNIX-like ope-rating systems and creates bot-net from them [23].

To form a set of harmless files, it was decided to take ELF files from the Linux OS, located in the /usr/bin directory, and the MD5 sum of which was not found in the VirusTotal virus database. Thus, it was possible to obtain a set of 959 harmless ELF files.

After data collection, we received a dataset consisting of 3958 binary ELF files. The dataset contains 959 harmless files, 980 Mirai-type threats, 995 Gafgyt-type threats, and 1024 Lightaidra-type threats. The proportion of file types in the created dataset is shown in Fig. 2.

The next step of the research was the disassembly and section-by-section reading of the binary files. For the section of each of the files, assembly commands were sequentially read, each of which was formed into a line of the form (6)

$$c\_str = (opcode, op\_str), \qquad (6)$$

where c_str is command row; $opcode$ is operation code; op_str is operands row.

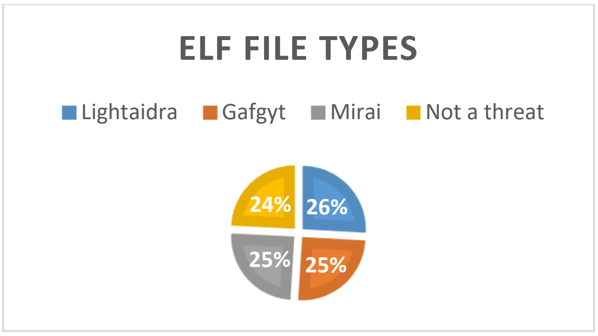
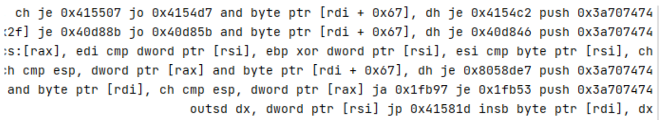


Fig. 2. **Pie chart for proportion of file types in the created dataset**
***Source*: compiled by the authors**

After that, special newline characters were replaced by spaces in the lines. A set of file's parsed assembly commands is shown in Fig. 3.



```
 ch je 0x415507 jo 0x4154d7 and byte ptr [rdi + 0x67], dh je 0x4154c2 push 0x3a707474
:2f] je 0x40d88b jo 0x40d85b and byte ptr [rdi + 0x67], dh je 0x40d846 push 0x3a707474
:s:[rax], edi cmp dword ptr [rsi], ebp xor dword ptr [rsi], esi cmp byte ptr [rsi], ch
:h cmp esp, dword ptr [rax] and byte ptr [rdi + 0x67], dh je 0x8058de7 push 0x3a707474
 and byte ptr [rdi], ch cmp esp, dword ptr [rax] ja 0x1fb97 je 0x1fb53 push 0x3a707474
                         outsd dx, dword ptr [rsi] jp 0x41581d insb byte ptr [rdi], dx
```

Fig. 3. **Set of file's parsed assembly commands**
Source: compiled by the authors

Having received a set of assembly commands for each of the files, we can proceed to their vectorization and training of machine learning models.

TRAINING AND EVALUATING CLASSIFICATION MODELS

The processed input data was passed to the input to the next step - n-gram generation. The resulting n-grams were submitted for vectorization using the TF-IDF algorithm. Using TF-IDF, the list of assembler commands from program files was vectorized and transformed into a sparse matrix of dimension <*n*, *n_features*>, where *n* – amount of documents, *n_features* – amount of significant features, chosen by the algorithm.

The sparse matrix, obtained after TF-IDF vectorization of the assembly command dataset, was submitted for classification. The dataset was divided into training and testing partitions: 80 % for training and 20% for testing. The time spent on vectorizing the input data and dividing it into train and test sets was 0.4 seconds.

To evaluate the results of each of the methods, the average classification accuracy and the weighted F1 value will be calculated [24].

The average classification accuracy will be calculated as the proportion of correctly predicted classes to the total number of predictions.

To calculate weighted F1, you must first calculate Precision and Recall (7-8).

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad (7)$$

where TP is amount of true positive predictions of the class; FP is amount of false positive predictions of the class.

$$\text{Recall} = \frac{TP}{TP + FN}, \qquad (8)$$

where TP is amount of true positive predictions of the class; FN is amount of false negative predictions of the class.

After calculating precision and recall, you can calculate F1 (9).

$$F1 = 2 * \frac{\text{Precision*Recall}}{\text{Precision+Recall}}, \qquad (9)$$

where Precision is precision of classification; Recall is recall of classification.

The weighted F1 is calculated for all classes of the sample, taking into account the proportion of the number of instances of each class relative to all instances of the sample, which is defined as the weight of the class (10) [24].

$$\text{F1}_{weighted} = \sum_{i=1}^{N} \text{F1}_i * W_i, \qquad (10)$$

where N is amount of classes; F1_i is F1 for class i; W_i is weight of the class i.

To determine the basic threshold of classification, Multinomial Naïve Bayes was trained and evaluated for all possible program sections and the corresponding number of n for n-gram. Assessment results of the basic threshold of classification are shown in Table 1. After obtaining the results, n-gram count and file section was selected based on the the highest values of the average accuracy and F1 weighted.

To improve the results, Laplace smoothing was used in the Bayesian classifier. This method allows to avoid obtaining zero probabilities of the word appearing in the text by adding a parameter to the calculated probability α [26]. Using hyper parameter tuning, it was established α=0.01.

Based on the best obtained mean accuracy and $F1_{weighted}$, was selected $n-gram=4$ and section $rodata$. For the Multinomial Naïve Bayes classifier learning curves were plotted (Fig. 4).

For all of the models in the paper, we build learning curves using the same approach. Selected model is initialized on chosen hyper parameters and cross-validated on the training dataset, which is divided into the train and the test sets. As an evaluation metrics, accuracy score is used.

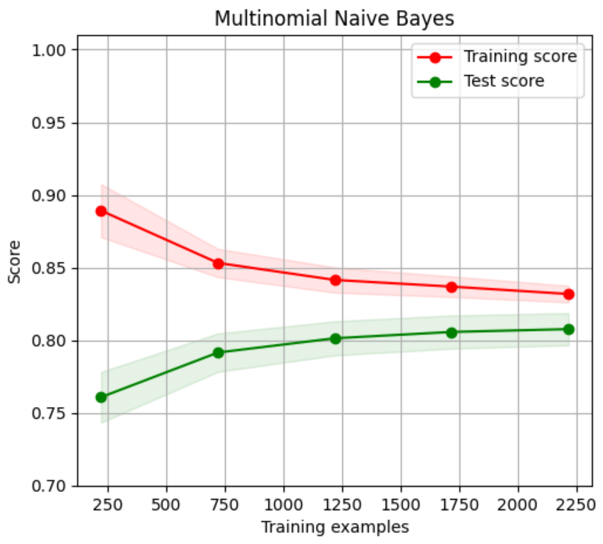


Fig. 4. **Learning curves for the Multinomial Naïve Bayes classifier**
***Source:* compiled by the authors**

Table 1. **Assessment of the basic threshold of classification based on n-gram count and file section**

Model	Size of n-gram	File section	Mean accuracy	Value of $\text{F1}_{weighted}$
Multinomial naive Bayes (α=0.01)	1	text	0.53	0.43
		data	0.54	0.45
		rodata	0.77	0.75
		bss	0.46	0.29
Multinomial naive Bayes (α=0.01)	2	text	0.53	0.43
		data	0.61	0.56
		rodata	0.79	0.78
		bss	0.46	0.29
Multinomial naive Bayes (α=0.01)	3	text	0.53	0.44
		data	0.66	0.63
		rodata	0.79	0.78
		bss	0.46	0.29
Multinomial naive Bayes (α=0.01)	4	text	0.55	0.46
		data	0.68	0.66
		rodata	0.82	0.82
		bss	0.46	0.29
Multinomial naive Bayes (α=0.01)	5	text	0.55	0.47
		data	0.67	0.65
		rodata	0.81	0.81
		bss	0.46	0.29

***Source*: compiled by the authors**

Learning curves for Multinomial Naïve Bayes model indicate the change in model accuracy with changing sample size and show convergence with increasing sample size. The curves are approximate for the real training sample size. This may indicate that a we achieved bias-variance tradeoff as a result of model training. It also gives a reason to claim that the model is not overtrained. We obtained $mean\_accuracy = 0.82$ and $F1_{weighted} = 0.82$, which is good and indicates the ability of the model to obtain enough information from the train sample and generalize on the test sample. With an increase in the sample, further convergence of the learning curves and an increase in the test accuracy of the model are possible.

To estimate the amount of computation spent on training the model, a graph of the average time in seconds spent on training versus the size of the training sample was plotted. The resulting graph for the model is shown in Fig. 5.

As the sample size increases, the training time increases linearly.

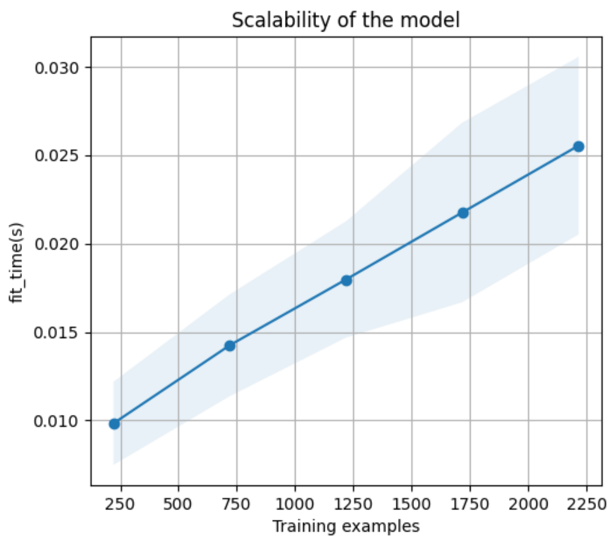


Fig. 5. **Plot of the average time in seconds spent on training the model, depending on the size of the training sample for Multinomial Naïve Bayes model**
Source: **compiled by the authors**

In order to visually evaluate the best detected classes of binary files from the test sample, a confuse-onmatrix was constructed, shown in Fig. 6. After analy-zing it, we can see that not a virus program files and viruses Lightaidra were predicted the most accurately by the Multinomial Naïve Bayes classifier.

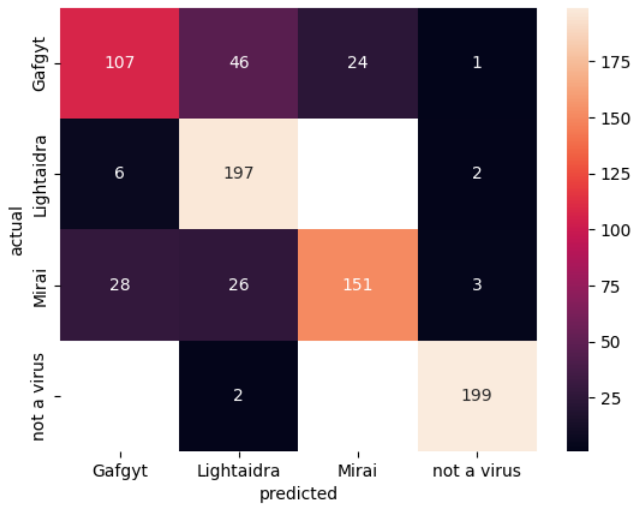


Fig. 6. **Confusion matrix for Multinomial Naïve Bayes classifier**
Source: **compiled by the authors**

After obtaining the classification threshold, it was decided to use the grid search algorithm to find the best hyperparameters for the support vector machine model (SVM). The GridSearchCV algorithm cross-validates the model on the Cartesian product of hyperparameters values and selects the best set of hyperparameters based on the highest obtained accuracy of the model [27].

Support vector machine (SVM) hyperparameters were analyzed:

– C – a measure of regularization. At low values of C, the hyperplane has a large offset to the points, at high values the offset is minimal. Set of investigated values: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

– kernel – a function that accepts data and transforms them into the required form, changing their dimensionality. Three functions are used: linear (linear), polynomial (poly), radial basis (rbf) and sigmoid (sigmoid).

The graph, that shows cross-validated average score obtained by the model, depending on the values of hyper parameters, is shown on Fig. 7. After analyzing the graph, it was determined that the highest score on the training data is obtained for hyper parameters $C = 0{,}9$ and kernel=rbf .

To assess the ability of the model to learn and to generalize, learning curves were built. Resulting curves are shown in Fig. 8.

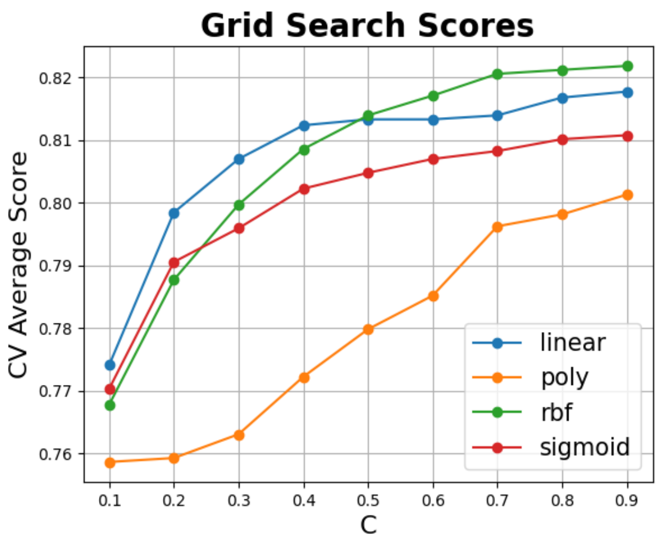


Fig. 7. **Graph of the average accuracy of the model depending on the hyper parameter *C* and *kernel* values for SVM model**
***Source:* compiled by the authors**

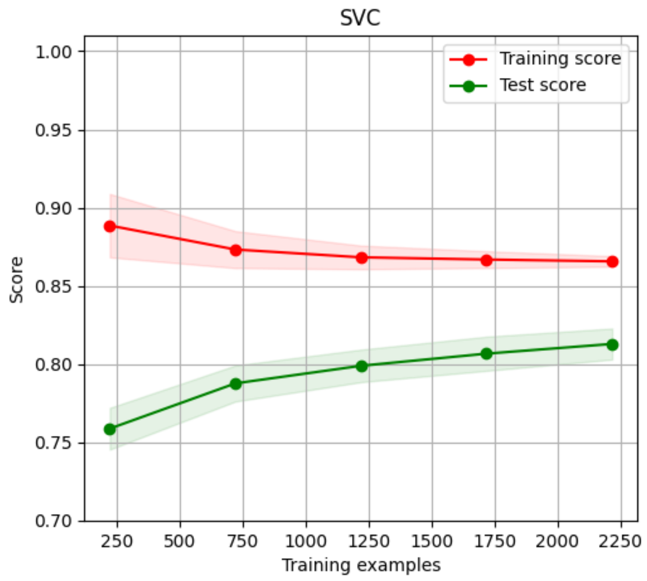


Fig. 8. **Learning curves for SVM model**
***Source:* compiled by the authors**

It can be seen from the graph that the accuracy of predictions on the train and test data converge with the increase of the sample, but do not converge on the final value. This may indicate that with an increase the size of the train dataset, the accuracy on the test data may also increase. A gap between the training and test curves may indicate the presence of variance. However, the accuracy of the test is not too low. SVM model with selected hyperparameters trained on the test data showed $mean\_accuracy = 0.83$ and $F1_{weighted} = 0.82$.

Confusion matrix for SVM models is shown on Fig. 9.

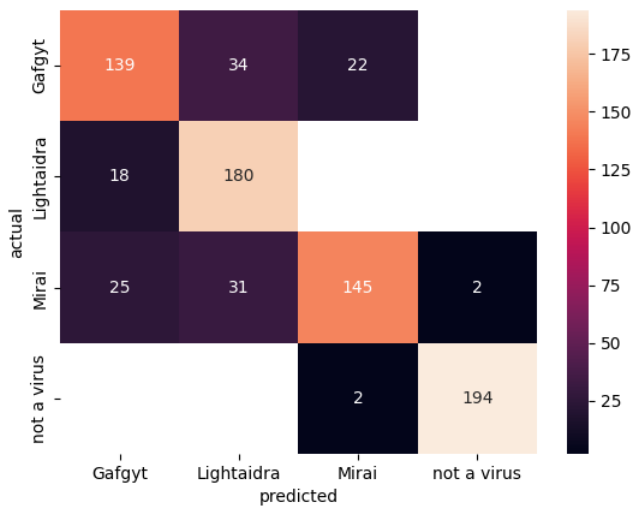


Fig. 9. **Confusion matrix for SVM model**
***Source:* compiled by the authors**

A plot of the average time in seconds spent on training the model, depending on the size of the training sample is shown in Fig. 10.

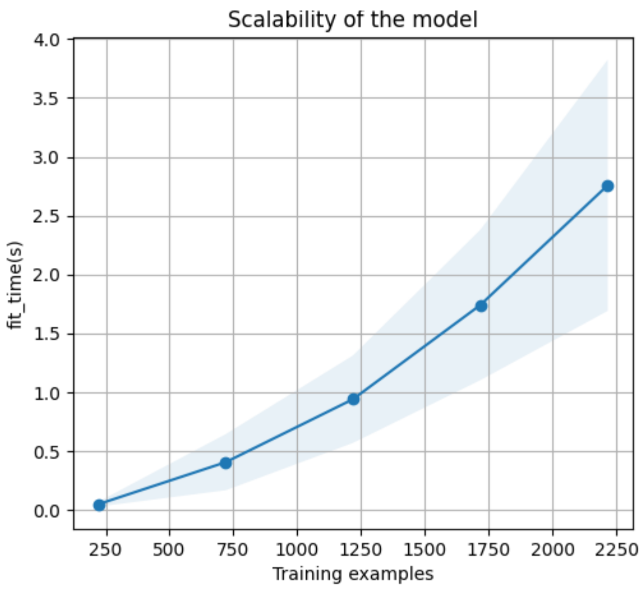


Fig. 10. **Plot of the average time in seconds spent on training the model, depending on the size of the training sample for SVM model**
***Source:* compiled by the authors**

To improve the prediction results, a stochastic gradient descent (SGD) model was built based on the support vector model.

The following hyper parameters were analyzed for the stochastic gradient descent model:

– eta0 – the initial value of the learning rate. A parameter used to calculate the learning rate;

– learning_rate (lr) – learning rate calculation algorithm.

Tested algorithms: "constant" is calculated according to the formula (11); "optimal" – the algorithm proposed by Leon Botto in the work [25]; "invscale" according to formula (12); "adaptive" – an algorithm according to which the learning rate is divided by 5, upon reaching a loss value that does not change for 5 iterations in a row.

$$lr_{cons} = eta0\,, \tag{11}$$

where $eta0$ is initial value for learning rate.

$$lr_{inv} = eta0 \;/\; \sqrt{t}\,, \tag{12}$$

where t is value of the element from the training sample.

The graph of the dependence of the accuracy of the model on the hyperparameter values is shown in Fig. 11. The graph shows that the highest accuracy was obtained for $lr_{optimal}$ та $eta0 = 0{,}01$.

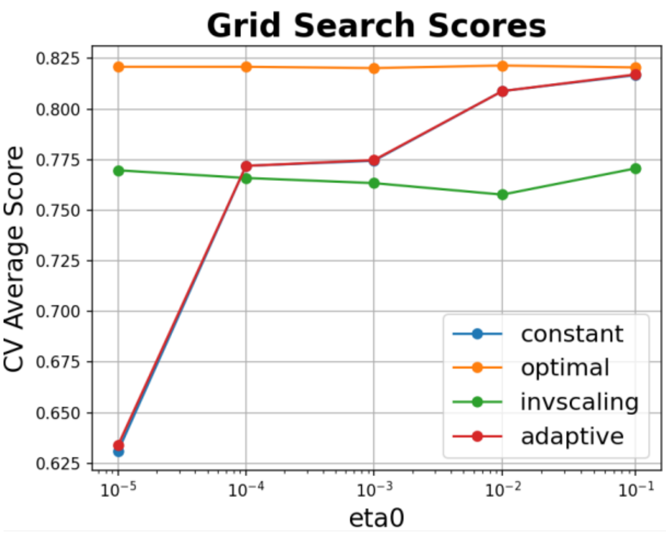


Fig. 11. **Graph of the average accuracy of the model depending on the hyper parameter *lr* and *eta0* values for SGD model**
Source: **compiled by the authors**

Learning curves were constructed for the model with the selected hyperparameters, which are shown in Fig. 12. The figure shows that the curves for training and test accuracy tend to converge, but do not coincide. A similar result was obtained for the model of support vectors. We can conclude that with an increase in the sample, the convergence of the curves is possible, however, on this sample, the variance is noticeable after training the stochastic gradient descent.

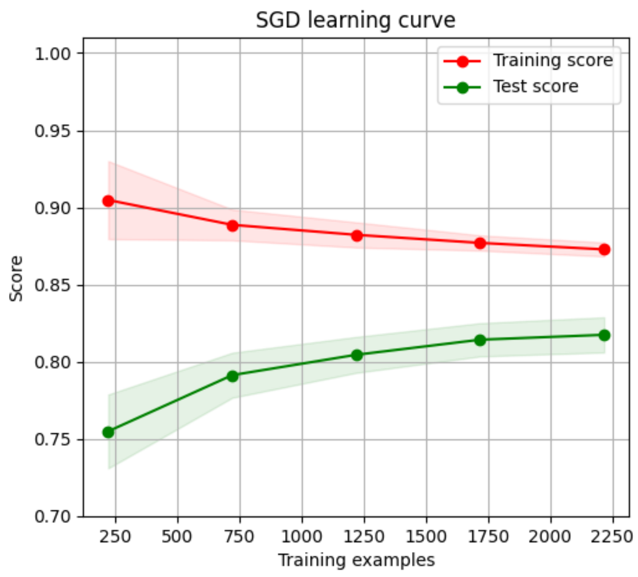


Fig. 12. **Learning curves for SGD model**
Source: **compiled by the authors**

The graph of the dependence of the average training time of SGD model on the sample size is shown in Fig. 13.

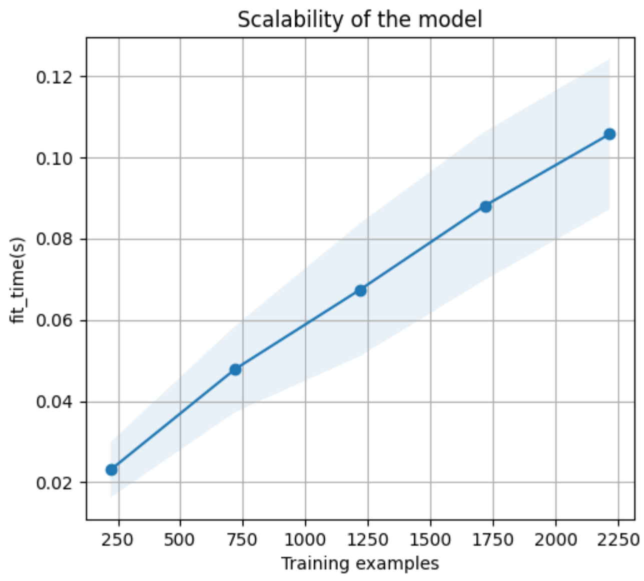


Fig. 13. **Plot of the average time in seconds spent on training the model, depending on the size of the training sample for SGD model**
Source: **compiled by the authors**

On the test sample, the model of support vectors using stochastic gradient descent showed $mean\_accuracy = 0.84$ and $F1_{weighted} = 0.84$. The results improved by 1 %, showing a slight optimization obtained by stochastic gradient descent.

The confusion matrix for the support vector method with stochastic gradient descent is shown in Fig. 14.

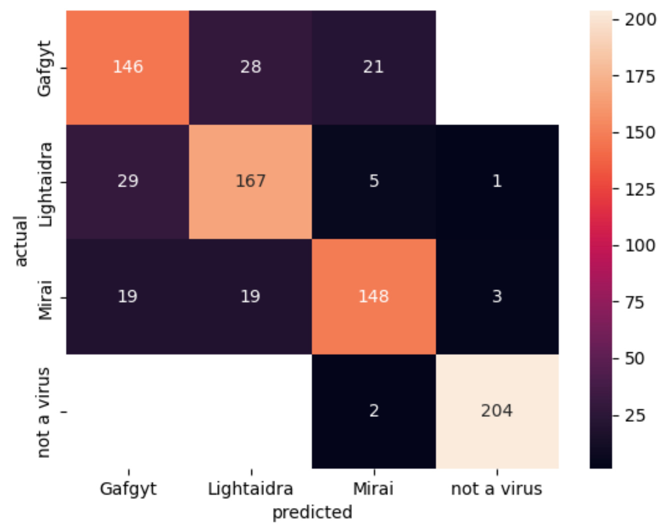


Fig. 14. **Confusion matrix for SGD model**
Source: **compiled by the authors**

To complement the evaluation of the results obtained using SVM and SGD, classification was carried out using gradient boosting model XGBoost. As boosters, decision tree models were chosen, the ensemble of which is used to select the most optimal model.

The following hyperparameters were analyzed to build the gradient boosting model:

– max_depth – the maximum depth of the decision tree.

– learning_rate (lr) – learning speed.

The graph of the dependence of the average accuracy of the model on the hyperparameter values is shown in Fig. 15. The best accuracy indicators for the model were obtained for $lr = 0.1$ and max_depth=7.

To check for overtraining and variance, the learning curves shown in Fig. 16 were constructed.

After analyzing the learning curves, we see that they tend to converge, but do not converge at the end of the sample. The same behavior was observed for previously studied models. We can conclude that the presence of variance is observed on this sample, which can be eliminated by training on a larger sample. The model has no signs of overtraining, since the accuracy obtained on the test sample is not significantly lower than the accuracy obtained on the training data. Obtained test results for gradient boosting models $mean\_accuracy = 0.83$ and $F1_{weighted} = 0.83$.

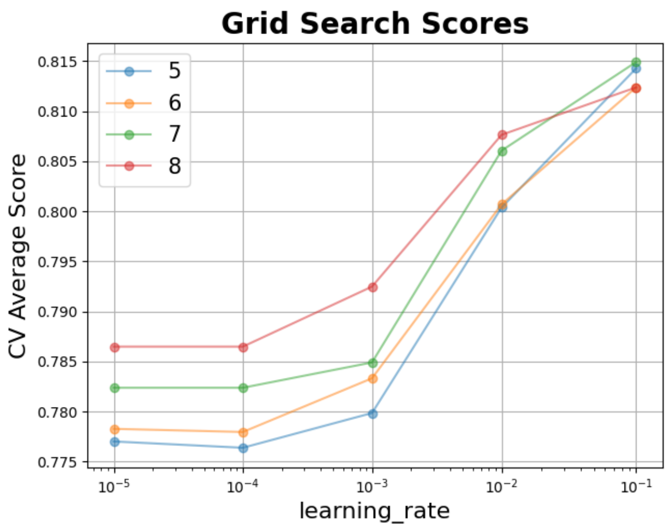


Fig. 15. **Graph of the average accuracy of the model depending on the hyper parameter *lr* and *max_depth* values for XGBoost model**
Source: **compiled by the authors**

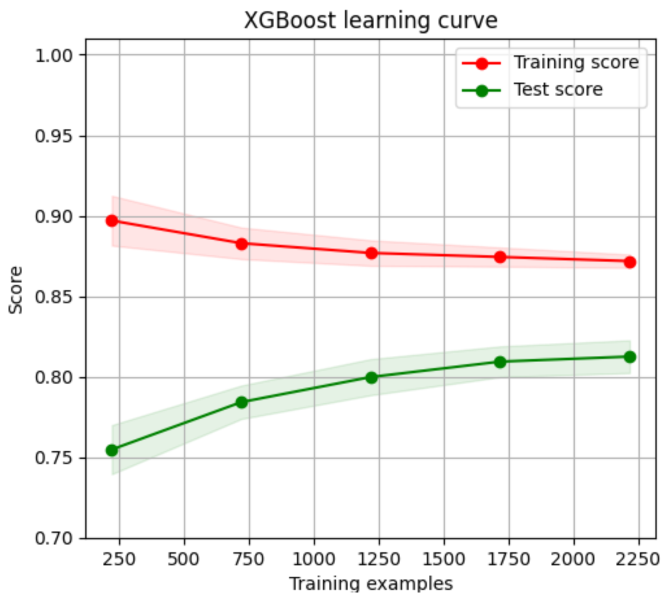


Fig. 16. **Learning curves for XGBoost model**
Source: **compiled by the authors**

The confusion matrix for gradient boosting model is shown in Fig. 17.

The dependence of the time spent on training gradient boosting models on the sample size is shown in Fig. 18.

To compare the obtained results, a table was created, in which it is given for each classifier

$mean\_accuracy$, $F1_{weighted}$ and the average time spent training on the test sample, which can be obtained from the graphs.

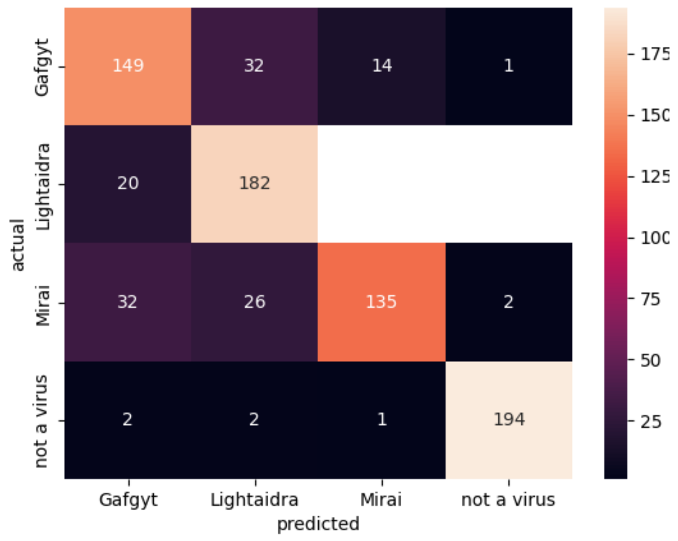


Fig. 17. **Confusion matrix for XGBoost model**

Source: **compiled by the authors**

The time specified in Table 2 does not include the time spent on vectorization of input data and division into training and test samples, which was 0.4 seconds.

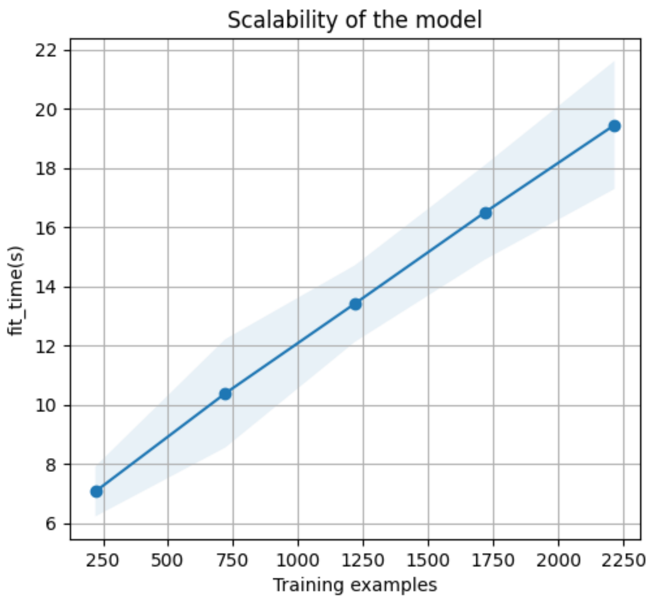


Fig. 18. **Plot of the average time in seconds spent on training the model, depending on the size of the training sample for XGBoost model**

Source: **compiled by the authors**

Table 2. **Malware and not a virus ELF files classification results for MultinomialNB, SVM, SGD and XGBoost**

Model	Mean accuracy (percent)	Value of $F1_{weighted}$ (percent)	Fit time (seconds)
MultinomialNB	82	82	0.026
SVM	83	82	2.75
SGD	84	84	0.1
XGBoost	83	83	19.5

Source**: compiled by the authors**

DISCUSSION OF THE RESULTS

The files collected and typed by the ELF antivirus software were used as a basis for creating an algorithm for identifying malicious software. The resulting data set was supplemented with safe ELF software files from the /usr/bin directory. The analysis of the distribution of classes indicated that the dataset for training machine learning models is balanced, which had a positive effect on the classification accuracy.

As a result of the comparison of the basic classification threshold for the semantically processed program sections, section rodata and n-gram size 4, which showed the best average accuracy and F1, were selected. The number of n-gram 4 can be explained by the fact that it is close to the number of words in the line of the assembly command according to formula 6 and, thus, highlights the semantically important fragments of the assembly code. The best classification performance for the rodata section can be explained by the fact that it contains initialized data, such as text variables, the values of which can carry significant features for the created classification model.

Obtained training results of the machine learning models showed relatively good average accuracy and weighted F1. The basic classification threshold of 82 %, obtained for the Multinomial Naive Bayes model, was improved, obtaining a maximum value on the test sample of 84% for the Stochastic Gradient Descent model, which indicates the correct choice of machine learning models. It should be noted that the time spent on training the support vector model using stochastic gradient descent is 0.1 seconds, compared to the support vector model without stochastic gradient descent – 2.75 seconds. This is due to the stochastically selected instances at each iteration of the training to calculate the weights, compared to using the entire data set for a model without stochastic gradient descent. The longest training time was shown by the gradient boosting model, which is explained by the use of an ensemble of models to solve the classification problem. The best time shown by the polynomial Bayesian classifier is due to the speed of calculations for probabilistic models.

CONCLUSIONS

As a result of the work, we solved the tasks of creating a set of malicious and safe binary ELF files; disassembly and vectorization of assembly commands of files, and classification of files according to their labels using machine learning models.

Obtained results of file classification by four different machine learning models confirm the expediency of using each of the models to solve the given problem.

The best result of 84 % accuracy on the test data was shown by the stochastic gradient descent model. The study of the learning curves of this model showed the possibility of improving the results with an increase in the training sample. Model training takes 0.1 seconds on a training sample size of 2250 files. This indicator allows you to use the model to detect threats in real time, as well as to train it on expanded volumes of data.

Based on the obtained results, it can be concluded that the methods for detecting malicious and safe software for UNIX-like operating systems were extended by the developed algorithm. The effectiveness of the developed algorithm was confirmed by various metrics, in particular the obtained classification accuracy. This confirms the achievement of the research goal.

FUTURE WORK

Possible steps to improve the obtained results may be to expand the set of input data. The learning curves for the selected stochastic gradient descent model show the ability to improve prediction accuracy with the increasing training samples.

Also, as a future work, the automation of the created algorithm is considered by creating an API and connecting to a database for recognizing malicious software in real time. The short training time for the gradient stochastic descent model allows it to be trained on new data in real time.

REFERENCES

1. Lazo, E. G. ”Combing through the fuzz: Using fuzzy hashing and deep learning to counter malware detection evasion techniques”. Microsoft. 2021. – Available from: https://www.microsoft.com/en-us/security/blog/2021/07/27/combing-through-the-fuzz-using-fuzzy-hashing-and-deep-learning-to-counter-malware-detection-evasion-techniques/ – [Accessed: Nov. 2021]

2. Huang, C. & Karnik, A. “The rise of deep learning for detection and classification of malware”. McAfee, 2021. – Available from: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/the-rise-of-deep-learning-for-detection-and-classification-of-malware/ – [Accessed: Nov. 2021].

3. Yusirwan, S., Prayudi, Y. & Riadi, I.. “Implementation of malware analysis using static and dynamic analysis method.” *International Journal of Computer Applications (0975–8887)*. 2015; 117 (6): 11–15. DOI: https://doi.org/10.5120/20557-2943 .

4. Shiel, I. & O'Shaughnessy, S. “Improving file-level fuzzy hashes for malware variant classification”. *Digital Investigation*. 28. https://www.scopus.com/authid/detail.uri?origin=resultslist& authorId=57191184033. 2019; 28: S88–S94. DOI: https://doi.org/10.1016/j.diin.2019.01.018.

5. Raff, E., et al. “Malware detection by eating a whole exe.” *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018. DOI: https://doi.org/10.48550/arXiv.1710.09435.

6. Zhong, F., Zekai, C., Minghui X., Guoming Z., Dongxiao Y. & Xiuzhen C. “Malware-on-the-Brain: Illuminating malware byte codes with images for malware classification”. *IEEE Transactions on Computers*. https://www.scopus.com/authid/detail.uri?authorId=57219436659. 2022. DOI: https://doi.org/10.48550/ arXiv.2108.04314.

7. Kang, B., Yerima, S.Y., Mclaughlin & Sezer S. “N-opcode analysis for android malware classification and categorization”. *International Conference On Cyber Security And Protection Of Digital*

Services (Cyber Security). https://www.scopus.com/authid/detail.uri?authorId=35728940000. 2016. p. 1–7. 7502343 DOI: https://doi.org/10.1109/CyberSecPODS.2016.7502343.

8. Qin, B., Junpeng, Z. & Honguy, C. “Malware detection based on TF-(IDF&ICF) method”. *Journal of Physics: Conference Series.* 2021; 2024 (1). DOI: https://doi.org/10.1088/1742-6596/2024/1/012030.

9. Cozzi, E., Graziano, M., Fratantonio, Y. & Balzarotti, D. "Understanding Linux Malware." *2018 IEEE Symposium on Security and Privacy (SP).* https://www.scopus.com/authid/detail.uri?origin=resultslist&authorId=57203246428. 2018. p. 161–175, DOI: https://doi.org/10.1109/SP.2018.00054.

10. “Portable formats specification”. Tool Interface Standards. *TIS Committee.* 1993.

11. Jurafsky, D. & Martin, J. H.”N-gram language models”. *Speech and Language Processing.* 2021; Chapter 3, Draft.

12. Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J. & Scholkopf, B. “Support vector machines”. *IEEE Intelligent Systems*. 1998. p. 18–28. DOI: https://doi.org/10.1109/5254.708428 .

13. Aatila, M., Mohamed, L. & Kartit A. “An Overview of gradient descent algorithm optimization in machine learning: Application in the ophthalmology field”. *Communications in Computer and Information Science.* https://www.scopus.com/authid/detail.uri?authorId=57203053740. 2020. p. 349–359. DOI: http://doi.org/10.1007/978-3-030-45183-7_27.

14. Schaetti, N. “UniNE at CLEF 2017: TF-IDF and deep-learning for author profiling”. Conference: *CLEF, Dublin.* 2017. DOI: https://doi.org/10.13140/RG.2.2.14902.60482.

15. Lei, Y., Hu, T. & Tang, K. “Generalization performance of multi-pass stochastic gradient descent with convex loss functions.” *Journal of Machine Learning Research.* https://www.scopus.com/authid/detail.uri?authorId=53663866600. 2021; 22 (25): 1–41.

16. Kibriya, A. M., Frank, E., Pfahringer, B. & Holmes, G. “Multinomial naive bayes for text categorization revisited”. *Australasian Joint Conference on Artificial Intelligence. AI 2004: Advances in Artificial Intelligence.* https://www.scopus.com/authid/detail.uri?authorId=8646743300. 2004. p. 488–499. DOI: https://doi.org/10.1007/978-3-540-30549-1_43.

17. Friedman, J. “Stochastic gradient boosting” *Computational Statistics & Data Analysis.* 2002; 38 (4): 367–378. DOI: https://doi.org/10.1016/S0167-9473(01)00065-2.

18. Tianqi, C. & Guestrin, C. “XGBoost: A scalable tree boosting system”. *Proceedings of the 22nd {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining.* https://www.scopus.com/authid/detail.uri?authorId=55788261800. 2016. p. 785–794. DOI: https://doi.org/10.1145/2939672.2939785.

19. “All new ELF binaries collected since the previous release in 2019”. VirusShare, 2020. – Available from: https://virusshare.com/torrents. – [Accessed: Dec. 2021].

20. “Get a file report”. VirusTotal, 2020. – Available from: https://developers.virustotal.com/reference/file-info. – [Accessed: Dec, 2021].

21. Margolis, J., Oh, T. T., Jadhav, S., Kim, Y. H. & Kim J. N. “An in-depth analysis of the mirai botnet”. *2017 International Conference on Software Security and Assurance (ICSSA).* 2017. p. 6–12, DOI: https://doi.org/10.1109/ICSSA.2017.12.

22. Sahota, J. & Vlajic, N. “Mozi IoT malware and its botnets: From theory To Real-World Observations”. *2021 International Conference on Computational Science and Computational Intelligence, CSCI.* https://www.scopus.com/authid/detail.uri?authorId=57795696900. 2021; p. 698–703. DOI: https://doi.org/10.1109/ CSCI54926.2021.00181.

23. McNulty, L. & Vassilakis, V. “IoT botnets: Characteristics, exploits, attack capabilities, and targets.” *13th International Symposium on Communication Systems, Networks and Digital Signal Processing. CSNDSP.* https://www.scopus.com/authid/detail.uri?authorId=57939684100. 2022. p. 350–355. DOI: https://doi.org/10.1109/ CSNDSP54353.2022.9908039.

24. Grandini, M., Bagli, E. & Visani G. “Metrics for multi-class classification: an overview”. 2020. DOI: https://doi.org/10.48550/arXiv.2008.05756 .

25. Bottou, L. “Stochastic gradient descent tricks” *Neural Networks: Tricks of the Trade.* https://www.scopus.com/authid/detail.uri?authorId=6701721644. 2012. p. 421–436. DOI: https://doi.org/10.1007/ 978-3-642-35289-8_25.

26. Alfons, J. & Hermann, N. “Reversing and smoothing the multinomial naive bayes text classifer” Pattern *Recognition in Information Systems. Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems, PRIS 2002. In conjunction with ICEIS 2002. Ciudad Real.* 2002. p. 200–212.

27. Adnan, M., Alarood, A., Uddin, M. I., & Rehman, I. “Utilizing grid search cross-validation with adaptive boosting for augmenting performance of machine learning models.” *PeerJ Computer Science.* https://www.scopus.com/authid/detail.uri?authorId=57202148561. 2022; 8. e803. DOI: https://doi.org/10.7717/ peerj-cs.803.

28. Su, Jiang & Shirab, Jelber. “Large Scale Text Classification using Semisupervised Multinomial Naive Bayes” *Proceedings of the 28th International Conference on Machine Learning, ICML 2011.* Bellevue, Washington: USA. 2011.

Conflicts of Interest: The authors declare that there is no conflict of interest

Received 25.10. 2022
Received after revision 15.12.2022
Accepted 22.12.2022

DOI: https://doi.org/10.15276/aait.05.2022.25
УДК 004.89

Семантичний аналіз і класифікація шкідливого програмного забезпечення для UNIX-подібних систем з використанням методів машинного навчання

Міщенко Максим Валерійович
ORCID: https://orcid.org/0000-0001-9769-9759; it144111@stu.cn.ua
Дорош Марія Сергіївна
ORCID: https://orcid.org/0000-0001-6537-8957; mariyaya5536@gmail.com. Scopus Author ID: 56912183600
Національний університет «Чернігівська політехніка» вул. Шевченка, 95. Чернігів, 14035, Україна

АНОТАЦІЯ

Стаття зосереджена на класифікації шкідливих програм на основі семантичного аналізу кодів операцій дизасембльованих секцій бінарних виконуваних файлів з використанням n-грам, індикатора TF-IDF і алгоритмів машинного навчання. Метою дослідження є вдосконалення та розширення наявних методів ідентифікації шкідливих програм, розроблених для UNIX-подібних операційних систем. Завданням дослідження є створення алгоритму, який може ідентифікувати типи загроз у шкідливих бінарних файлах для UNIX-подібних систем за допомогою n-грам, індикатора TF-IDF і алгоритмів машинного навчання. Процес класифікації шкідливих програм може базуватися на статичних або динамічних сигнатурах. Статичні сигнатури можуть бути представлені у вигляді послідовностей байт-коду, двійкових інструкцій або імпортованих бібліотек. Динамічні сигнатури можна представити як послідовність дій шкідливого ПЗ. Ми будемо використовувати стратегію статичних сигнатур для семантичного аналізу та класифікації шкідливих програм. У цій статті ми будемо працювати з двійковими файлами ELF, які є найпоширенішим типом виконуваних файлів для UNIX-подібних операційних систем. Для цілей цього дослідження було зібрано набір даних із 2999 зразків шкідливих ELF файлів, використовуючи дані із сайтів VirusShare та VirusTotal, а також 959 нешкідливих програмних файлів з директорії /usr/bin в операційній системі Linux. Шкідливі файли представляють одне з 3 сімейств шкідливих програм: Gafgyt, Mirai та Lightaidra, які є поширеними загрозами для UNIX-подібних систем. У отриманому наборі даних для кожного ELF файлу було проставлено мітку відподвідно до його типу. Запропонований алгоритм класифікації складається з кількох етапів підготовки: дизасемблювання кожного бінарного ELF файлу із набору даних і семантична обробка та векторизація інструкцій зі кожної з секцій файлу. Для встановлення порогу класифікації використовується поліноміальна модель Байєса. Використовуючи поріг класифікації, ми визначаємо розмір n-грам і секцію файлу, які дадуть найкращі результати класифікації. В результаті було виявлено, що найкраща точність класифікації отримана для n-gram розміру 4 та секції rodata.

Щоб отримати найкращу точність, буде використано декілька моделей машинного навчання разом із оптимізацією гіперпараметрів. Як метрика точності розробленого алгоритму використовується середня точність і зважена оцінка F1. Стохастичний градієнтний спуск для моделі SVM було обрано як найкращу модель ML на основі отриманих експериментальних результатів. Експериментально підтверджено ефективність розробленого алгоритму для класифікації шкідливих програм для UNIX-подібних операційних систем. Результати були проаналізовані та використані для висновків та пропозицій для подальшої роботи.

Ключові слова: Виявлення шкідливого програмного забезпечення; машинне навчання; семантичний аналіз; багатокласова класифікація; інтелектуальний аналіз тексту ; операційна система

ABOUT THE AUTHORS

Maksym V. Mishchenko - Postgraduate, Information Technology and Software Engineering Department. Chernihiv Polytechnic National University, 95, Shevchenko Street. Chernihiv, 14035, Ukraine
ORCID: https://orcid.org/0000-0001-9769-9759; it144111@stu.cn.ua
Research field: Cybersecurity; machine learning; operating systems; software engineering

Міщенко Максим Валерійович - аспірант, кафедра Інформаційних технологій та програмної інженерії. Національний університет «Чернігівська політехніка» вул. Шевченка, 95. Чернігів, 14035, Україна

Mariia S. Dorosh - Doctor of Engineering Sciences, Professor of Information Technology and Software Engineering Department, Chernihiv Polytechnic National University, 95, Shevchenko Str. Chernihiv, 14035, Ukraine
ORCID: https://orcid.org/0000-0001-6537-8957; mariyaya5536@gmail.com. Scopus Author ID: 56912183600
Research field: Modeling and design of intelligent systems; knowledge management; convergence of project management systems; human factor in information security systems of organizations and projects

Дорош Марія Сергіївна - доктор технічних наук, професор кафедри Інформаційних технологій та програмної інженерії. Національний університет «Чернігівська політехніка», вул. Шевченка, 95. Чернігів, 14035, Україна