UDC 004.62

Comparative analysis of machine learning methods for detecting malicious files

Alan Nafiiev[1], Hlib Kholodulkin[2], Andrii Rodionov[1]

[1] *National Technical University of Ukraine «Igor Sykorsky Kiev Polytechnic Institute», Institute of Physics and Technology*

[2] *Taras Shevchenko National University of Kyiv, Faculty of Computer Science and Cybernetics*

Abstract

Nowadays, one of the most critical cyber security problems is the fight against malicious software, precisely, the problem of detecting it. Every year, new modern computer viruses are created that are capable of mutation and changing while running. But unfortunately, the developers of antivirus software do not have time to quickly add all types of malicious programs to the signature databases. In this regard, it is sensible to use heuristic detection methods based on algorithms of machine learning. The purpose of this paper is to present several classification methods based on machine learning techniques for detecting zero-day attacks. In particular, the following algorithms were tested: random forest classifier, support vector classifier, greed search in svc, and k-nearest neighbours. The dataset was taken from the Kaggle website. It consists of 19611 executable files of the PE format, 14599 of which are malicious, and 5012 files are benign. This article presents recommended classification and detection methods with advanced analysis of important metrics that allow you to assess and compare machine learning algorithms' effectiveness and performance for detecting malware.

Keywords: intrusion detection, malware detection, PE format, machine learning, zero-day, malware classifiers

Introduction

Historically, the first antivirus programs used a signature-based approach to detect malicious files. Signatures are features of a malicious application that characterize its code and method of intrusion. The essence of this approach is to check the contents of the analyzed object for the presence of signatures of already known threats. However, methods based on the comparison of instances are static. Although they have a high accuracy in determining the fact of an attack, they can identify only a limited part of all possible threats. An obvious drawback is an impossibility of detecting non-trivial, polymorphic, and unknown malicious programs, the signatures of which have not yet been identified. Therefore, this analysis cannot be used to detect zero-day attacks. An alternative approach is to use the following existing methods: behavioral analysis, machine learning, deep learning, etc. [1].

The idea of this work is similar to that of Faranak Abri et al. [2], where extensive research was conducted on various machine learning methods to detect zero-day attacks. However, our work includes several significant additions. Namely, to determine the accuracy of the investigated methods and analyze the result, we will use not only the accuracy metric but also the F-score. We will also consider errors of the 1st and second levels, which will allow for a deeper analysis of the results. Additionally, we will compare the methods in terms of the speed of operation and the optimal use in conditions of a limited data set size.

This article explores the effectiveness of classical machine learning methods. The aim is to empirically validate machine learning models to detect malicious files and possible zero-day attacks.

The paper is organized as follows: the first section discusses the main research issues and general information about the applied machine learning algorithms. Section 2 describes the research dataset. Ways for evaluating methods are outlined in section 3. The classifiers and their main parameters are presented in section 4. Section 5 compares the results of several machine learning methods. Section 6 describes the possibility of using the methods on a real system. Observations from the analysis of experimental data are presented in the conclusions section at the end of this article.

1. The main observed problems

This article describes experiments that demonstrate the possibility of using models based on machine learning methods to detect malicious programs and zero-day attacks without prior knowledge of these threats. The main task of machine learning methods is to find the connection between the analysed data and the detected signs of harmfulness. Many different ML methods are used to determine the connection, starting with Conventional Machine Learning and ending with neural networks, both simple (with one or a pair of hidden layers) and more complex, called deep learning.

It is assumed that deep learning methods cope with the classification task better than classical ML algorithms [3]. However, this is not always possible. In addition, complex neural networks require significant computing resources and large datasets to work correctly, which may not always be optimal for use on a

real system. Therefore, in this article, we will look at the work of four classical machine learning algorithms:

The random forest classifier was chosen as one of the most statistically productive. The method also works great with a mixture of numeric and categorical functions, leading to good results. The random forest combines two main ideas: using an ensemble of decision trees and the random subspace method. The algorithm does not operate with metrics, allowing you to work with features of a different nature freely. Also, this method is very stable when working with sparse and uncalibrated data.

Support Vector Machines have been taken as one of the most popular algorithms. According to numerous studies [4, 5], it shows promising results precisely in problems of binary classification. It is based on the search for a separating hyperplane with a maximum gap in this space. In classical SVC, based on kernels, a vector of weights is trained, which determines the contribution of each sample to the dividing hyperplane between classes by solving an optimization problem.

The GridSearch method used for SVC has been chosen as one of the most traditional functional methods used to optimize the loss function. The grid search algorithm cyclically selects groups of C and g values and computes the model accuracy for each pair. As a result, we obtain (C, g), which provides the highest accuracy.

Also, we used the method of k-nearest neighbours – the most straightforward metric algorithm for classifying objects. The K-NN is based on the following rule: an object belongs to the class to which most of its nearest neighbors belong. It is worth noting that K-NN does a poor job with large data because with a large number of subsets, the complexity of calculating the distances between sets increases. The research described in this article focuses on the following tasks:

1) Conduct several experiments to investigate various machine learning models for detecting malicious software.
2) Conduct a study of algorithms using a wide range of indicators in a limited and extended dataset.
3) Find out whether at least one of the classification methods used can work on a real system and what is required for this.

2. Input Datasets

For the input, the set of executable files from the Kaggle website are used. According to the information on the website, the raw dataset contained 14,599 files marked as «legitimate» and 5012 files marked as «malicious». The labels were «1» and «0», respectively. The number of features is reported as – 77. Table 1 contains the features in the model.

Experiments are conducted on three groups of data of different sizes, which will allow us to evaluate the behavior of algorithms with a decrease in the training sample and an increase in the test set, respectively. The models are trained on the following data: 90 % for the first group, 70 % for the second, and 40 % of the main dataset for the third group, respectively. The model trained on the first group is tested on "unlabelled" data containing 1962 files. The second and third groups are tested on samples containing 5884 and 11767 files, respectively.

Table 1. PE features

Features	
e_magic	MinorImageVersion
e_cblp	MajorSubsystemVersion
e_cp	MinorSubsystemVersion
e_crlc	SizeOfHeaders
e_cparhdr	CheckSum
e_minalloc	SizeOfImage
e_maxalloc	Subsystem
e_ss	DllCharacteristics
e_sp	SizeOfStackReserve
e_csum	SizeOfStackCommit
e_ip	SizeOfHeapReserve
e_cs	SizeOfHeapCommit
e_lfarlc	LoaderFlags
e_ovno	NumberOfRvaAndSizes
e_oemid	SuspiciousImportFunctions
e_oeminfo	SuspiciousNameSection
e_lfanew	SectionsLength
Machine	SectionMinEntropy
NumberOfSections	SectionMaxEntropy
TimeDateStamp	SectionMinRawsize
PointerToSymbolTable	SectionMaxRawsize
NumberOfSymbols	SectionMinVirtualsize
SizeOfOptionalHeader	SectionMaxVirtualsize
Characteristics	SectionMaxPhysical
Magic	SectionMinPhysical
MajorLinkerVersion	SectionMaxVirtual
MinorLinkerVersion	SectionMinVirtual
SizeOfCode	SectionMaxPointerData
SizeOfInitializedData	SectionMinPointerData
Size Of UnintializedData	SectionMaxChar
AddressOfEntryPoint	SectionMainChar
BaseOfCode	DirectoryEntryImport
ImageBase	DirectoryEntryImportSize
SectionAlignment	DirectoryEntryExport
FileAlignment	ImageDirectoryEntryExport
MajorOperatingSystemVersion	ImageDirectoryEntryImport
MinorOperatingSystemVersion	ImageDirectoryEntryResource
MajorImageVersion	ImageDirectoryEntrySecurity

For data standardization, we use StandardScaler function from Python library sklearn. StandardScaler implies equating the mean of each feature in the dataset to zero. This is especially useful for methods like K-NN and SVC. Then the principal component analysis (PCA) is used to reduce the dimension of the input data.

3. Accuracy assessments methods

As an indicator for the accuracy of our models, we will use not only the standard accuracy metric but also F1-score, which is calculated from precision and recall. This metric will provide us with more representative results. Formulas for computing all the metrics are represented in equations (1), (2), (3), (4). Moreover, the confusion_matrix function is used to assess the accuracy. The function can be found in Python's library sklearn. The function is used to calculate the confusion matrix is displayed in Fig. 1. Where TP, TN, FP, and FN represent True Positives, True Negatives, False Positive, and False Negatives, respectively. Based on these four parameters, type I and type II errors are calculated. Type I error – is a false-positive result, i.e., we falsely suppose that a file is malicious. Type II error occurs when we mark a malicious file as legitimate.

$$precision = \frac{TP}{TP + FP} \tag{1}$$

$$recall = \frac{TP}{TP + FN} \quad (2)$$

$$F1 = \frac{2 * precision * recall}{precision + recall} \quad (3)$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP} \quad (4)$$

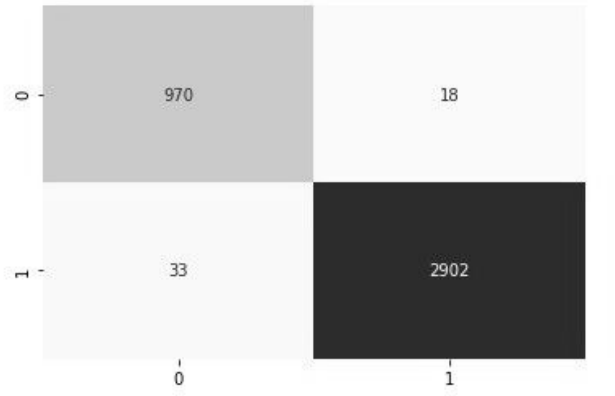

Fig. 1. Confusion matrix

4. Explored algorithms

4.1. K-nearest neighbors algorithm

K - nearest neighbors algorithm implements the qualification via KNeighborsClassifier function. The algorithm is based on calculating similarity scores between two objects. In our model, the similarity between two points x and y is calculated via Minkowski metric:

$$p(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^1/p \quad (5)$$

For building the model, the following parameters are used:

KNeighborsClasifier(algorithm='ball_tree', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=18, p=2, weights='uniform')

Parameter «uniform» means that all the weights are equal. Parameter «ball_tree» means that distances between objects are stored in a trees structure, which increases the speed of finding the closest neighbors. «leaf_size» – the threshold for activation of full screening if we used «BallTree» for finding neighbors. The larger the leaf_size, the faster the tree is built since fewer nodes need to be created. Minkowski metric with parameters «p» and «n_neighbors» are optimal quantities of the nearest neighbors, which have been found empirically.

4.2. Support vector machine method

Support vector machine method maps initial data to the space with more dimensions, which allows building an optimal separating hyperplane with the maximum clearance in this dimension [6]. SVC method in multidimensional space uses Kernel of a radial basis function, which lets us optimally separate non-linear data into a linear one.

The kernel function RBF finds the similarity between two objects via the following equation:

$$K(x, y) = exp(-\lambda||x - y||^2) \quad (6)$$

The principle of the kernel is visualized in Fig. 2.

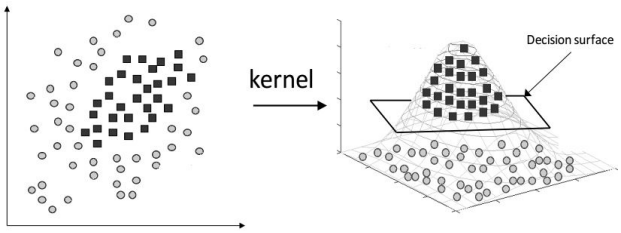

Fig. 2. How kernel works

The SVC function implements the classification, where the vector of weights "α" is trained based on kernels. The vector of weights determines the contribution of each sample to the separating hyperplane between classes by solving the following optimization problem:

$$(\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n}\alpha_i) = S_k(\alpha) \quad (7)$$

subject to the constraints:

$$\sum_{i=1}^{n}\alpha_i y_i = 0, \quad (8)$$

$$0 \leqslant \alpha_i \leqslant C \quad (9)$$

For building the model the following parameters are used:

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='rbf', max_iter=-1, probability=True, radom_state=None, shrinking=True, tol=0.00, verbose=False)

Coefficient «C» – is the parameter that allows adjusting the relationship between maximizing the separating bandwidth and minimizing the total error. Parameter «gamma» is the «bandwidth» of the RBF kernel. It is involved in constructing the model and can cause overfitting. In our case, this parameter is selected automatically based on the number of features.

4.3. Random forest classifier

Random forest algorithm is implemented via RandomForestClassifier, which creates a set of decision trees from randomly selected subsets of the training sample.

Table 2. The Python packages and parameter settings

Classifier	Type	Python Package	Parameter Set & Values
k-Nearest Neighbors	Supervised Learning	KNeighborsClassifier(...)	n_neighbors=17, metric='minkowski'
SVC	Supervised Learning	SVC(...)	probability = True
Random forest classifier	Ensemble Learners	RandomForestClassifier(...)	n_estimators = 30, max_depth = 6
Greed in SVC	Supervised Learning	GridSearchCV(...)	param_grid = {'C':[50,100], 'gamma':[0.1]}

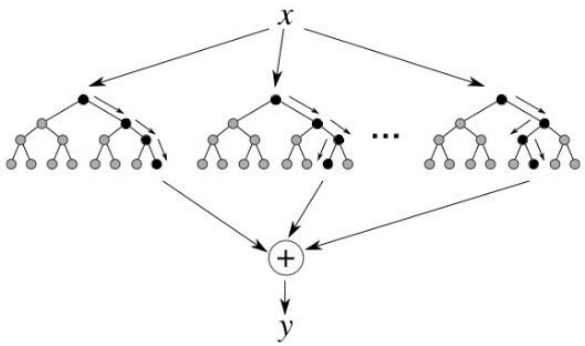

Fig. 3. Construction of random forest

It takes into account all the results from the trees to classify the test object. Final RF-classifier «$\alpha(z_{rf})$» decides by the majority of votes found by decision trees. [7]:

$$\alpha(z_{rf}) = sign(\sum_{j=1}^{r} b(z_{rf}^{j})) \qquad (10)$$

Where $b(z_{rf}^{j})$ – the solution of the base classifier of $j-th$ tree ($j = \overline{1,r}$), and z_{rf}^{j} – random subsample.

Constructing of random forest is demonstrated in Fig. 3. For building the model the following parameters are used:

RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini', max_depth=6,
max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=30,
n_jobs=None, oob_score=True, random_state=0,
verbose=0, warm_start=False)

Parameter n_estimators – number of trees. The more trees – the better quality, but the learning and computing time increases. max_depth – maximum depth of trees. The less the depth, the faster RFC builds and functions. With the increase in depth, the quality of learning dramatically increases as well [8]. The rest of the parameters have default values.

4.4. Gridsearch

Functional method Gridsearch is used to optimize the loss function of SVC method [9]. The grid search algorithm cyclically selects groups of C and g values and calculates model accuracy for each pair. It then prints out the pair (C, g) that provides the highest precision. The following parameters are used to build our model:

GridSearchCV(cv='warn',
error_score='raise-deprecating',
estimator=SVC(C=1.0, cache_size=200,
class_weight=None, coef0=0.0,
decision_function_shape='ovr', degree=3,
gamma='auto_deprecated', kernel='rbf',
max_iter=-1, probability=True, random_state=None,
shrinking=True, tol=0.001, verbose=False),
iid='warn', n_jobs=None, param_grid='C': [50, 100],
'gamma': [0.1], pre_dispatch='2*n_jobs', refit=True,
return_train_score=False, scoring=None, verbose=3)

«param-grid» – dictionary of hyperparameters, where we supply selected values, based on which the GridSearchCV training will be conducted. We have to train ten models to assess SVC's efficiency for specific values of C and gamma via a five-block cross-validation check. The main drawback of cross-validation checks is the time needed to train all the models. Parameter estimator is a model to be trained. The rest of the parameters have default values.

5. Results

According to Table 3, all our explored methods of classification have a good accuracy. To make sure of this, let us analyze another metric – F1 score.

Just as has been expected, the best result demonstrated the ensemble method of random forest. The high percentage of F1-score – 93 %, insignificant type I and type II errors, and the short time for training give an exciting prospect of using this method for solving real systems. Also, note that GridSearch method for SVC slightly improved the results of basic SVC but dramatically slowed down the execution speed, which makes this method less attractive to use. In spite of its simplicity, K-nearest neighbors algorithm also showed a decent result in terms of precision and time of training. This result is justified by the optimally selected number of neighbors ($n = 17$).

Let us pay attention to type I and type II errors. Note that in our case, not detecting a malicious file is more dangerous than marking a «legitimate» file as malicious. Except for the SVC method, all of our studied classification methods have a type II error within 2 %. It has given the worst performance in this regard, with an error of about 5 %.

We also note that with an increase in the volume of the test sample and a proportional decrease in the training sample, all the studied methods behave similarly: the accuracy slightly decreases, and, accordingly, the training time of the model decreases.

Table 3. Results

Trainset	Metrics	Random forest classifier	SVC	Greed in SVC	K Nearest Neighbors
90%	Accuracy	96.32%	91.23%	94.92%	94.33 %
	F1-score	92.78%	82.28%	90.23%	90.98 %
	Type 1 error	1.70%	3.56%	2.93%	2.62 %
	Type 2 error	1.98%	5.20%	2.14%	2.03 %
	Time	721 ms	5.20%	4min 20s	42.8 ms
70%	Accuracy	95.83%	91.00%	93.92%	94.02%
	F1-score	91.73%	81.82%	88.24%	88.23%
	Type 1 error	1.70%	3.70%	3.32%	2.82%
	Type 2 error	2.47%	5.29%	2.75%	3.14%
	Time	413 ms	9.45 s	55.7 s	22.2 ms
40%	Accuracy	95.49%	90.61%	94.31%	93.69%
	F1-score	92.13%	81.09%	87.78%	87.49%
	Type 1 error	1.97%	4.38%	3.06%	2.98%
	Type 2 error	2.53%	4.99%	2.61%	3.32%
	Time	389 ms	6.12 s	45.1 s	9.7 ms

6. Using Random Forest in real system

After analyzing the results of our methods, it can be noted that the random forest method has shown good efficiency and has good prospects for its use for detecting malicious programs on real systems.

Based on the experimental data obtained, it is possible to determine the throughput of the model (the number of files divided by the time spent on training the model). According to table 3, the RFC model is capable of training 25 thousand files in one second, which is a good result.

The dataset taken from the Kaggle website is sufficient to cover many different types of malware relevant to the modern world. However, it does not cover all possible types. Before launching a project, the model should be trained on the maximum number of different types of malware in an actual application. It is also necessary to investigate the problems of choosing various essential features that describe the file, both numerical and categorical.

Conclusions

After analyzing the results of the experiments, it can be argued that the random forest is applicable to the detection of malicious files. It showed the best result: 96 % is the accuracy metric, and almost 93 % is the F-score. The main limitation of the method is that the use of a large number of trees significantly affects the computational speed of the algorithm, which is ineffective for real-time prediction. Therefore, should optimally select all the parameters for building the model. In most natural systems, the random forest algorithm is fast enough, but situations may arise when the speed and performance at runtime are low. Then it is preferable to use other faster methods without loss of performance in detection accuracy. It is also worth noting that the k-nearest neighbor's method can well be used on a natural system with a good choice of parameters and relevant input data. It showed the accuracy result in a couple of percent less than the random forest algorithm: 94 % accuracy metric and 91 % – F-score.

Also, the analysis of the performed experiments has demonstrated that the efficiency of the conventional support vector machine can be significantly improved with the help of the selection of hyperparameters along the grid. Accuracy has increased from 91 % to 94 %. Such an operation entails an increase in the operating time of the method by about eight times. This problem can be solved by using enormous hardware computing resources to run the algorithm.

References

[1] S. Sharma and C. R. Krishna, "Zero-day malware detection based on supervised learning algorithms of api call signatures," Australasian Data Mining Conference (AusDM11), December 2011.

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

[3] D. Gavrilu, M. Cimpoeu, and D. A. L. Ciortuz, "Malware detection using machine learning," International Multi-conference on Computer Science and Information Technology (IMCSIT), 2009.

[4] Caruana, "An empirical comparison of supervised learning algorithms," 2006.

[5] M. Ginny, "The implementation of support vector machines using the sequential minimal optimization algorithm," 2000.

[6] Y. Lifshits, "Algorithms for internet: Support vector machines," 2006.

[7] L. A. Demidova and I. A. Klueva, "Random forest algorithm in the problem of improving the quality of svm-classification," 2011.

[8] R. Genuer, J.-M. Poggi, C. Tuleau-Malot, and N. Villa-Vialaneix, "Random forests for big data," 2017.

[9] G. Xie, Y. Zhao, S. Xie, M. Huang, and Y. Zhang, "Multi-classification method for determining coastal

water quality based on svm with grid search and knn," *[J]. Int J Performability Eng*, 2019.