AN ENSEMBLE-BASED MALWARE DETECTION MODEL USING MINIMUM FEATURE SET

Eslam Amer[1,2,✉] and Ivan Zelinka[1]

[1]Technical University of Ostrava, Czech Republic
[2]Misr International University, Egypt
eslam.amer@vsb.cz✉, ivan.zelinka@vsb.cz

Abstract

Current commercial antivirus detection engines still rely on signature-based methods. However, with the huge increase in the number of new malware, current detection methods become not suitable. In this paper, we introduce a malware detection model based on ensemble learning. The model is trained using the minimum number of signification features that are extracted from the file header. Evaluations show that the ensemble models slightly outperform individual classification models. Experimental evaluations show that our model can predict unseen malware with an accuracy rate of 0.998 and with a false positive rate of 0.002. The paper also includes a comparison between the performance of the proposed model and with different machine learning techniques. We are emphasizing the use of machine learning based approaches to replace conventional signature-based methods.

Keywords: *malware detection, machine learning, ensemble learning.*

Received: 13 September 2019
Accepted: 08 December 2019
Published: 21 December 2019

1 Introduction

Cybersecurity threats and security breaches are growing rapidly nowadays. Security breachers are fluctuating to the new tools and opportunities in cyberspace. The rapid growth rate in security threats is also coupled with an increasing rate in social networks developments. Social networks are considered as rich environments for spreading threats more frequently. Malicious programs (malware) are considered one of the most ever-growing threats. Malware authors also are making use of software vulnerabilities and improper software security configuration. They are continually developing sophisticated methods to hide malware from being detected. Anti-malware and similar detection tools have to continuously develop new tools in order to cope with the continual developments of malware threats.

Current malware detection techniques can be categorized into signature and non-signature approaches [1]. Signature-based approaches are commonly used by commercial antivirus software. Signatures are binary patterns that are extracted from malware samples [1]. The accuracy rate for signature-based approaches is high. However, the accuracy is limited only to the known identifiable malware signatures. Therefore, the antivirus signature database should be regularly updated to cope with the newly discovered signatures. Another primary drawback is that signature-based approaches considered inefficient against unknown attacks such as "zero-day attack". Unknown vulnerabilities allow an attack window time, that is the time between threat discovery to signature update. It is shown in [2-6] that malware authors use tools to pack and obfuscate malware to hide malware and to escape from detection. Consequently, signature-based approaches are inefficient to cope with malware authors tricks and therefore, they are unreliable.

Non-signature based approaches identify malware by inspecting their behaviors. Through analyzing malware behaviors, specific behavioral patterns could be extracted. The extracted patterns form the malware behavioral models. Non-signature approaches detect malware by comparing malicious behavior to already predefined models. Malware may have different names, however, they share some similar behaviors. The detection strategy is based on the behavioral similarity between the unknown malware to the indexed stored behaviors. In this way, the detection of unknown malware is possible. The behavioral analysis could be viewed as a replacement for static signature-based methods. However, non-signature based approaches can be vulnerable to false positives or false negatives results. Technically, it identifies normal benign files as malware and incapable to detect malware in the latter case [5-7]. However, it totally avoids the attack window time.

Machine Learning (ML) provides a possible solution to overcome the drawbacks of signature-based solutions. Through innovating a learning framework, it becomes possible to process unseen malware samples. The learning models are trained using a set of features that are extracted from analyzed malware. Malware

features may include various attributes of software modules, and machine code instructions. Ensemble learning is an effective branch of machine learning that is used to improve the accuracy and performance of traditional machine learning classifiers [8]. It works by creating a core group of learners and combining their outputs for final decision making. Ensemble learning take advantage of complementary information of different classifier to improve the performance and accuracy of the decision.

In this paper, we introduce a non-signature based approach. Our approach is aiming to identify and detect malware based on extracting integrated feature set that describe the malware. The features are obtained through extracting significant features from the malware Portable executable (PE) header. The proposed model utilizes different learning techniques to produce an ensemble learning model. All techniques are trained using the most significant features extracted from the PE header. The proposed ensemble model is used to identify whether a given sample is benign or malware. Our contributions in this paper include:

1. Implementation on an efficient malware detection model based on ensemble learning.
2. Evaluation of the efficiency for different classifiers in characterizing and predicting malware.

This paper is organized as follows: section 2 describes the related works, section 3 introduces the experimental design, results and discussions are present in sections 4, and finally, section 5 presents our conclusions.

2 Related Works

In this section, we are going to discuss some of the previous works that applied machine learning in malware analysis. Machine learning has been successfully applied to the identification and detection of malware [9,10]. It was shown in [11] that machine learning can also be used to characterize malware families. The expression malware analysis refers to the study of malware's behavior and how to detect and eliminate it. The objective behind malware analysis is to understand how malware work. The analysis requires extracting significant features that will be used for automatic identification of malware. Knowing how malware works can help in building better defenses against malware attacks. In general, malware features can be gathered through either static or dynamic analysis of malware samples [9].

Static analysis is the process of analyzing and extracting features from the meta-data of the binary file without executing it. The extracted features include specific string patterns, byte sequences, operation codes, and library calls. Those features are used to determine whether a file is malicious or not [12]. However, static analysis approaches on their own are insufficient. The reason is due to obfuscation methodologies that are widely used by malware authors which makes static approaches unreliable [9,13]. Furthermore, pattern matching approaches are considered effective in known malware patterns. However, it is considered ineffective, and hence unreliable for unknown or new patterns such as zero-day or polymorphic malware attacks.

In contrast to static analysis, dynamic analysis requires the executable file to be running to aggregate generated outcomes for feature extraction [9,11]. Dynamic analysis is proposed to capture the dynamic behavior for packed or obfuscated malware. The analysis mainly is done in a secured virtual environment or sandbox. The analysis intended to monitor the real-time execution of malware specifically; its malicious code. The analysis of dynamic behavior involves extracting features such as system calls, registry change, memory usage, and network behavior [14].

The main advantage of dynamic analysis is that it reveals how the malware behaves. Those real behaviors are not accessible in static analysis. Therefore, most dynamic analysis research works rely on real-time behavioral patterns. It is shown that patterns such as API-call sequence as well as control flow as major features that capture malware behavior [15,16]. However, dynamic analysis approaches are also imperfect. It is reported in [3, 5-9] that smart malware can detect whether it runs on a virtual or real environment. Moreover, smart malware can modify their behavior by hiding their malicious code to avoid detection. Although their imperfection, dynamic analysis is prospectively able to grasp some benchmark metrics. Those metrics are discovered during malware interactions with the underlying system. Such metrics can be utilized to detect a possible attack [17].

Malware detection is considered a complex classification problem because malware may adapt escape techniques to avoid detection. The elasticity of machine learning made it possible to depend on any type of data to classify and detect malware. Saxe and Berlin [18] used static program features derived from the program code. The features are trained using deep feed-forward neural network to differentiate between malware and benign files with an accuracy of 97.45%. However, Grosse et al. [19] indicates that relying on static features could yield to non-reliable classification results. They pointed out that there are some issues like code obfuscation may negatively affect the whole classification performance. Evaluations in [19] showed that the classification performance is downgraded from 97% to 20% when classifying obfuscated codes. Bruce Ndibanje et al. [3] proposed a hybrid static/dynamic model for unpacking and de-obfuscating malware to make use of

static data. They also address the behavior analysis of malware by analyzing API call sequence that makes it possible to understand malware behavior.

Tian et al. [20] utilized the frequency of API-calls with Windows XP to classify malware samples. They applied several classification algorithms and achieved 97% accuracy ratio using Decision Table (DT) and Random Forest (RF). Damodaran et al. [21] used a hybrid static/dynamic model to classify malware. They rely on Hidden Markov Model (HMM) with API calls to classify a malware to malware families which achieve 98% accuracy score.

Fang et al. [22] proposed an approach to classify malware using ensemble learning algorithms. They rely on Term Frequency-Inverse Document Frequency (TF-IDF) as a feature selection method. TF-IDF selects the highly ranked features from high discriminative features in different scales of datasets. Zhang et al. [23] used ensemble learning to construct a lightweight malware classification system. The resulted classifier has the ability to classify malware into their corresponding family even if the training data is imbalanced.

Both static and dynamic approaches to malware analysis produce a set of features. Such features can be transformed into a characterization form which represents the characteristics of the analyzed files. The characterization form can be translated into detection patterns that are considered as worthy profit to machine learning models. It could also be used to realize the harmful intent of an attacker. In our work, we relied on Random forests as a feature selector to extract the most influenced features that could be used to distinguish benign and malware files. Our proposed model uses only the minimum set of features to create a classification pattern that has the ability to identify malware.

3 Experimental Design

The proposed work as described in Fig 1 is implemented in three main steps: feature selection, training phase, and testing phase. The model will be illustrated in details through the following subsections.

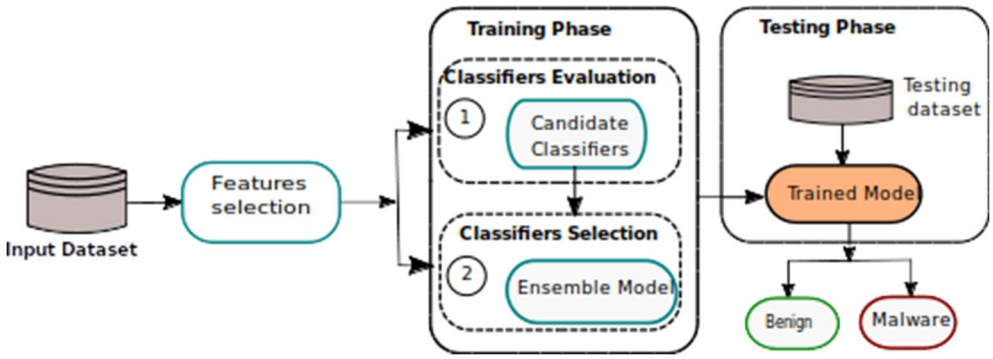

Figure 1: The proposed model

3.1 Dataset

The input dataset contains raw malware and benign samples that are collected from various sources. As explained in Table 1, our dataset contains 41324 benign records and 96724 malicious ones. Those records are collected from various Windows 7 executable files and VirusShare[1] respectively. Every sample record contains extensive static information found in each sample file. The dataset is comprised of 54 features which accounted for all information found in every sample record.

Table 1: Summary of created dataset

Type	Source	Quantity	Percentage
Malware	VirusShare	96724	70.1
Benign	Various windows programs	41323	29.9

1 https://virusshare.com/

3.2 Feature Selection

The objective of feature selection is to simplify our model by including only the most significant and relevant features. There are many approaches to select the most important features [24]. Selecting the most appropriate features will result in reducing the dimensionality of the raw dataset. Such reduction will reflect in return to the model variance, overfitting, and of course reducing the model computation cost.

In the proposed work, Random Forest is used as features selector. Random Forest is widely used as a feature selector technique [25, 26]. The reason behind that is due to the tree-based design implemented by random forest. The tree design is ranked by how to enhance node purity. Nodes are positioned in the tree levels based on their purity score. Nodes with a considerable increase in purity positioned at the start of the trees, while nodes with decreased purity positioned at the bottom level of the tree. Therefore, by pruning trees below a purity level, we gain subsets of the most influenced features.

3.3 Model Training

The training phase is considered to be the core step in the proposed model. The objective of the training is to evaluate the performance of different candidate learning models. In our work, we used eleven different classifiers that are belonging to different learning models to be our candidate classifiers. The training phase as shown in Fig 1 has two sub-phases namely classifiers evaluation, and classifiers selection respectively. In classifiers evaluation, all classifiers are trained using the same training set. The best top K classifiers are selected and combined into an ensemble model. The choice of K is depending on the learning accuracy threshold value. In our experiment, the threshold value $\alpha = 0.5$. In the classifiers selection phase, the formed ensemble model is also trained using the same training set as carried out in the classifiers evaluation phase. Performance of the ensemble model is also evaluated. The output model of ensemble training is representing our final training model.

In our experiment, the finial prediction decision of the ensemble model relied on voting classifier. There are two optional strategies for voting classifier which are hard voting or soft voting. In the hard voting option (known also as majority voting), the class selection process is depending only simple majority voting according to the accuracy [27]. However, in soft voting, the average is taken for the probability of each predicted class. The class selection is the one that is equivalent to the highest value. In our experiment, the ensemble model relies on a hard voting option.

3.3 Model Testing

Model testing is the last step of our proposed model. The testing involves evaluating the model using unseen malware and benign samples. The objective is to evaluate the model performance in terms of measuring the prediction accuracy in classifying new benign and malware samples. For the purpose of testing, we used 1000 benign and 1000 malware samples from the original dataset to be used as test samples.

4 Results and Discussion

In this section, we are going to present the experimental evaluations and findings of the proposed model. The evaluations include feature reduction, classifiers performance evaluations, and finally the proposed model performance evaluation.

4.1 Feature Reduction

As mentioned in section 3.1, feature reduction is a critical step to reduce the dimensionality of data and hold only the most significant features. In our experiment, the original number of features is 54. Experimental results showed that Random Forest selection algorithm marked only ten features to be significant. The selected features constitute only about 22% of all original features. The significant features extracted from PE headers are: *ResouresMinEntropy, VersionInformationSize, DllCharacteristics, SectionMaxEntropy, ResourceMaxEntropy, Machine, Characteristics, SubSystem, ImageBase, and SizeOfOptionalHeader*

The proposed model is trained using only the selected features. The other remaining features are discarded as they are considered insignificant. The reduction algorithm reduces the dataset by almost 78% of original data which in return decreases the complexity of our experiments.

4.2 Classifiers Performance Evaluation

In machine learning, it is required to build computational models that are able to generalize effectively the extracted features. When training a model, a disturbed generalization is recognized by over-training. A common way to avoid over-training is to use an appropriate data splitting.

Classification is defined as the process of finding a model or mapping function which separate data into multiple classes. In order to avoid the poor generalization, our model is experimented with different splitting ratios for training and testing.

Train-test-split

The objective of train/test experiments is to grasp and realize some patterns of data. Those patterns are used by the learning models to make almost accurate predictions. Based on experimental results, learning models could be judged according to their efficiency. The word *efficiency* means the capability of a learning model to generalize well to new data. Also, it was necessary to observe the performance of the different classifiers with the reduced features set.

To avoid the poor generalization problem, the candidate classifiers are experimented using two splitting criteria. Each experiment has a different train/test splitting ratio. The first experiment is done by splitting the reduced dataset into 80 / 20 for training and testing. Whereas, the second experiment is done by splitting the reduced dataset into 65 / 35 for training and testing respectively. Evaluation metrics such as accuracy, precision, recall, and F-measure are used to evaluate the candidate classifiers' performance. The calculations of evaluation metrics are shown in equations 1-4.

$$\text{Accuracy} = \frac{TruePositives(TP)+TrueNegatives(TN)}{TruePositives(TP)+TrueNegatives(TN)+FalsePositives(FP)+FalseNegatives(FN)} \tag{1}$$

$$\text{Precision} = \frac{TruePositives(TP)}{TruePositives(TP)+FalsePositives(FP)} \tag{2}$$

$$\text{Recall} = \frac{TruePositives(TP)}{TruePositives(TP)+FalseNegatives(FN)} \tag{3}$$

$$\text{F-measure} = \frac{2*Precesion*Recall}{Precision+Recall} \tag{4}$$

The results of classifiers' performance for the first and second splitting criteria are summarized in Table 2 and Table 3 respectively. Results showed that the classifiers performance showed almost typical results with some minor differences. The modification in splitting criteria hasn't influenced the classification decision. As shown in Table 2, and Table 3 we have implemented different binary classifiers like support vector machine (SVM), Naive Bayes (NB), K-nearest neighbour (KNN), Multi-layer perceptron (MLP), Linear Discriminate Analysis (LDA), and Decision Tree (DT). We also experimented some ensemble classifiers like Random Forests (RF), and ExtraTrees classifiers.

In terms of accuracy, Table 2 and Table 3 are showing that the Random Forest classifier returned the best result among all other classifiers. Other classifiers like Decision Tree, support vector machine, Multi-layer perceptron, and K-nearest neighbour also showed considerable high accurate results. Naive Bayes classifier showed intermediate results comparing to the aforementioned classifiers. However, Logistic Regression (LR) showed poor classification results compared to the other ones. The reason behind the poor performance of logistic regression is due to the imbalanced distribution of data in the training and test set that causes LR to cast everything off. Another reason might be due to the strong correlation among the features for each class.

According to our learning accuracy selection criteria, we have selected the top-5 classifiers to form our ensemble model. We haven't included any ensemble classifier in our initial combination of ensemble model even if they showed more accurate results. The reason for that is that we want to evaluate the accuracy of non-ensemble classifiers when they work together. However, we will experiment the combination of non-ensemble with ensemble classifiers to measure whether if it may affect the accuracy of our ensemble model.

Our proposed ensemble model will initially be formed with the following classifiers decision tree, linear discriminate analysis, Multi-layer perceptron, support vector machine, and K-nearest neighbour. As mentioned before, our ensemble model relied on the hard voting. In ensemble voting, the hard option used to select the output class based on the majority selections of classifiers.

Table 2: Candidate classifiers training result on train-test with splitting ratio (80 / 20)

Classifier	Accuracy	Precision		Recall		F1-score	
		Malware	Benign	Malware	Benign	Malware	Benign
XGB	0.988	0.99	0.98	0.99	0.98	0.99	0.98
ExtraTree Classifier	0.990	0.99	0.98	0.99	0.99	0.99	0.98
LDA	0.917	0.95	0.94	0.98	0.88	0.96	0.91
AdaBoost	0.984	0.99	0.98	0.99	0.98	0.99	0.98
Random Forest	0.994	0.99	0.99	1.00	0.99	1.00	0.99
Decision Tree	0.991	0.99	0.99	0.99	0.98	0.99	0.98
MLP	0.942	0.97	0.88	0.94	0.94	0.96	0.91
SVM	0.983	0.99	0.96	0.98	0.99	0.98	0.97
KNN	0.977	0.99	0.96	0.98	0.97	0.98	0.96
NB	0.701	0.70	1.00	1.00	0.00	0.82	0.00
LR	0.300	0.00	0.30	0.00	1.00	0.00	0.46

Table 3: Candidate classifiers training result on train-test with splitting ratio (65 / 35)

Classifier	Accuracy	Precision		Recall		F1-score	
		Malware	Benign	Malware	Benign	Malware	Benign
XGB	0.991	0.99	0.98	0.99	0.99	0.99	0.99
Extra Tree Classifier	0.989	0.99	0.98	0.99	0.98	0.99	0.98
LDA	0.945	0.95	0.94	0.98	0.87	0.96	0.91
AdaBoost	0.984	0.99	0.98	0.99	0.98	0.99	0.98
Random Forest	0.993	1.00	0.99	1.00	0.99	1.00	0.99
Decision Tree	0.990	0.99	0.98	0.99	0.98	0.99	0.98
MLP	0.965	0.97	0.96	0.98	0.94	0.93	0.94
SVM	0.968	0.99	0.96	0.98	0.99	0.98	0.97
KNN	0.976	0.99	0.96	0.98	0.97	0.98	0.96
NB	0.699	0.70	1.00	1.00	0.00	0.82	0.00
LR	0.300	0.00	0.30	0.00	1.00	0.00	0.46

4.2 Proposed Model Performance Evaluation

In order to verify the performance of the classification methods, we will use two additional evaluation metrics. The additional metrics are inspired by the confusion matrix which is a table that describes the performance of the classifier. The evaluation metrics are false positive rate (FPR), and false negative rate (FNR). The FNR is an indicating factor for the falsely predicted classes (equation 5), and FNR is indicating the incorrect negatively classified classes (equation 6).

$$\text{FPR} = \frac{FalsePositives(FP)}{FalsePositives(FP)+TrueNegatives(TN)} \tag{5}$$

$$\text{FNR} = \frac{FalseNegatives(FN)}{FalseNegatives(FN)+TruePositives(TP)} \tag{6}$$

The performance evaluation of the proposed model is carried out using four different experiments. The first experiment is implemented to evaluate the performance of each individual classifier. The second experiment is performed to evaluate the proposed ensemble model against individual classifiers. The third experiment evaluates the performance resulted when integrating two ensemble methods (proposed ensemble + random forests). The fourth experiment evaluates the combination of three ensemble methods (proposed ensemble + random forests + Extra tree). Table 4 showed that the performance accuracy of classifiers are relatively closed to each other with some minor differences. Random Forests and Decision Tree classifiers showed the best performance with high accuracy rate 0.999 and low false positive rate 0.001 comparing to other classifiers.

Evaluation of the proposed ensemble model is showed in Table 5. Experimentation showed that the performance of the proposed ensemble model is slightly better than the individual non-ensemble classifiers. The results demonstrated that the proposed ensemble model show a similar accuracy to the SVM classifier with rate of 0.994. We observed that the performance of the ensemble model showed a minor accuracy differences to the decision tree classifier that returned accuracy rate of 0.999. However, one of the most drawbacks of the decision tree is its vulnerability to the overfitting problem. The overfitting of the decision tree is recovered with the proposed ensemble. The reason is that ensembles are more generalized and unlikely to overfit [28].

Table 4: Performance evaluation of classifiers.

Classifier	Accuracy	False positive rate	False negative rate
XGB	0.995	0.010	0.000
Extra Tree Classifier	0.998	0.002	0.001
LDA	0.985	0.022	0.007
AdaBoost	0.995	0.012	0.000
Random Forests	0.999	0.001	0.000
Decision Tree	0.999	0.001	0.000
MLP	0.945	0.017	0.086
SVM	0.994	0.011	0.000
K-Nearest Neighbour	0.992	0.014	0.000

Table 6 shows the result of combining two ensemble methods. In this experiment, we evaluate the combination of the proposed ensemble model when it combined to another ensemble model like Random Forests. The combination of two ensemble learning models resulted in achieving more enhancements with more accuracy results 0.998 compared to the accuracy value of 0.994 obtained from the initial proposed ensemble model. However, we noticed from Table 4 and Table 5 that the performance of random forests works better than when it is not combined with other ensembles. Table 7 shows the result of combining three ensemble learning methods. In this experiment, we evaluate the combination of the proposed ensemble model when it combined to another ensemble models like Random Forests and Extra Tree Classifier. Results showed that the accuracy value returned from combing three ensemble learning methods is similar to accuracy returned when combing two ensemble learning methods. The combination of three ensemble models as shown in Table 7 is showing no difference at all between combinations of two ensemble models. We came to an experimental conclusion that the combination of multiple ensemble models can enhance the accuracy in general, however, the performance of some ensemble learning methods may be degraded when it combined with other ensemble methods.

The experimental evaluations are showing a promising path to solve the vulnerabilities that face non-signature based methods. We showed that false positives and false negatives can be decreased by using ensemble methods. The trained ensemble models can be used effectively as a replacement to signature-based engines. Anti-malware should rely on adaptive models to accommodate with the rapid increasing rate of malware.

Table 5: Performance evaluation of proposed ensemble model compared to other classifiers.

Classifier	Accuracy	False positive rate	False negative rate
Proposed Ensemble Model	**0.994**	**0.011**	**0.000**
XGB	0.995	0.010	0.000
Extra Tree Classifier	0.998	0.002	0.001
LDA	0.985	0.022	0.007
AdaBoost	0.995	0.012	0.000
Random Forest	0.999	0.001	0.000
Decision Tree	0.999	0.001	0.000
MLP	0.945	0.017	0.086
SVM	0.994	0.011	0.000
K-Nearest Neighbour	0.992	0.014	0.000

Table 6: Performance evaluation of combining two ensemble models compared to other classifiers.

Classifier	Accuracy	False positive rate	False negative rate
Proposed Ensemble Model + Random Forests	**0.998**	**0.002**	**0.000**
Proposed Ensemble Model	**0.994**	**0.011**	**0.000**
XGB	0.995	0.010	0.000
Extra Tree Classifier	0.998	0.002	0.001
LDA	0.985	0.022	0.007
AdaBoost	0.995	0.012	0.000
Random Forest	0.999	0.001	0.000
Decision Tree	0.999	0.001	0.000
MLP	0.945	0.017	0.086
SVM	0.994	0.011	0.000
K-Nearest Neighbour	0.992	0.014	0.000

Table 7: Performance evaluation of combining three ensemble models compared to other classifiers.

Classifier	Accuracy	False positive rate	False negative rate
Proposed Ensemble Model + Random Forests + Extra Tree Classifier	**0.998**	**0.002**	**0.000**
Proposed Ensemble Model + Random Forests	**0.998**	**0.002**	**0.000**
Proposed Ensemble Model	**0.994**	**0.011**	**0.000**
XGB	0.995	0.010	0.000
Extra Tree Classifier	0.998	0.002	0.001
LDA	0.985	0.022	0.007
AdaBoost	0.995	0.012	0.000
Random Forest	0.999	0.001	0.000
Decision Tree	0.999	0.001	0.000
MLP	0.945	0.017	0.086
SVM	0.994	0.011	0.000
K-Nearest Neighbour	0.992	0.014	0.000

5 Conclusion

In this paper, we have proposed an ensemble learning approach to overcome the drawbacks of current commercial signature-based approaches. Through reducing the data dimensionality, we were able to focus on the significant features that characterize malware PE files. Those features were experimented and evaluated by different classifiers. Experiments have shown that the classification accuracy was satisfactory. The paper also has shown experimental comparisons between the efficiency of different classifiers. Also, We have experimentally evaluated different ensemble learning models, we have shown that ensembles were more efficient than individual classifiers. Evaluations have shown that the proposed model has a great accuracy rate in identifying unseen malicious samples with an accuracy ratio of 0.998 and with false positive rates of 0.002. The paper has proposed a methodology to replace the current signature-based approach with a machine learning approach. It has been shown that machine learning approaches could be used with good training samples to overcome the current drawbacks of signature-based techniques. The rapidly increasing rate of unseen malware is considered as a curse to commercial antivirus software, however, it considers as a fortune for machine learning model. The more training with more malware training samples, the more accuracy, adaptation, tuning, and reliability of proposed machine learning models.

Acknowledgment. The following grants are acknowledged for the financial support for this research. Grant of SGS No. SP2019/137, VSB Technical University of Ostrava.

References

[1] Kumar, A., Kuppusamy, K. S., and Aghila, G. 2017. A learning model to detect maliciousness of portable executable using integrated feature set. *Journal of King Saud University-Computer and Information Sciences* 31, 2, pp. 252–265.

[2] Bahador, M. B., Abadi, M., and Tajoddin, A. 2019. HLMD: a signature-based approach to hardware-level behavioral malware detection and classification. *The Journal of Supercomputing* 75, 5551–5582.

[3] Ndibanje, B. et al. 2019. Cross-Method-Based Analysis and Classification of Malicious Behavior by API Calls Extraction. *Applied Sciences* 9, 2, 239. DOI: 10.3390/app9020239

[4] Alazab, M., Venkatraman, S., and Watters, P. 2009. Effective digital forensic analysis of the NTFS disk image. *Ubiquitous Computing and Communication Journal* 4, 1, pp. 551–558.

[5] Smith, M. et al. 2018. Dynamic Analysis of Executables to Detect and Characterize Malware. *17th IEEE International Conference on Machine Learning and Applications* (ICMLA). IEEE. DOI: 10.1109/ICMLA.2018.00011

[6] Yousefi-Azar, M. et al. 2018. Malytics: a malware detection scheme. *IEEE Access* 6, pp. 49418–49431.

[7] Rhode, M., Burnap, P., and Jones, K. 2018. Early-stage malware prediction using recurrent neural

networks. *Computers & Security* 77, pp. 578–594.
[8] Sayadi, H., Patel, N., Sasan, A., Rafatirad, S., and Homayoun, H. 2018. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *2018 55th ACM/ESDA/IEEE Design Automation Conference* (DAC). ACM. DOI: 10.1145/3195970.3196047
[9] Ucci, D., Aniello, L., and Baldoni, R. 2019. Survey of machine learning techniques for malware analysis. *Computers & Security* 81, pp. 123–147.
[10] Song, J. et al. 2017. Practical in-depth analysis of ids alerts for tracing and identifying potential attackers on darknet. *Sustainability* 9, 2, pp. 1–18.
[11] Kolosnjaji, B. et al. 2016. Deep learning for classification of malware system call sequences. *Australasian Joint Conference on Artificial Intelligence*. Springer, Cham, pp. 137–149.
[12] Gandotra, E., Bansal, D., and Sofat, S. 2014. Malware analysis and classification: A survey. *Journal of Information Security* 5, pp. 56–64. DOI: 10.4236/jis.2014.52006
[13] Burnap, P. et al. 2018. Malware classification using self organising feature maps and machine activity data. *Computers & Security* 73, pp. 399–410.
[14] Qiao, Y. et al. 2014. CBM: free, automatic malware analysis framework using API call sequences. *Knowledge engineering and management*. Springer, Berlin, Heidelberg, pp. 225–236.
[15] Luo, X. et al. 2016. An incremental-and-static-combined scheme for matrix-factorization-based collaborative filtering. *IEEE transactions on automation science and engineering* 13, 1, pp. 333–343.
[16] Zeng, N. et al. 2014. Image-based quantitative analysis of gold immunochromatographic strip via cellular neural network approach. *IEEE transactions on medical imaging* 33, 5, pp. 1129–1136.
[17] Ranveer, S. and Hiray, S. 2015. Comparative analysis of feature extraction methods of malware detection. *International Journal of Computer Applications* 120, 5, pp. 1–7.
[18] Saxe, J., and Berlin, K. 2015. Deep neural network based malware detection using two dimensional binary program features. *10th International Conference on Malicious and Unwanted Software* (MALWARE). IEEE, DOI: 10.1109/MALWARE.2015.7413680
[19] Grosse, K. et al. 2017. Adversarial examples for malware detection. *European Symposium on Research in Computer Security*. Springer, Cham, pp. 62–79.
[20] Tian, R. et al. 2010. Differentiating malware from cleanware using behavioural analysis. *5th international conference on malicious and unwanted software*. IEEE. DOI: 10.1109/MALWARE.2010.5665796
[21] Damodaran, A. et al. 2017. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* 13, 1, pp. 1–12.
[22] Fang, Y., Yu, B., Tang, Y., Liu, L., Lu, Z., Wang, Y., and Yang, Q. 2017. A new malware classification approach based on malware dynamic analysis. In *Australasian Conference on Information Security and Privacy*. Springer, Cham, pp. 173–189.
[23] Zhang, Y., Huang, Q., Ma, X., Yang, Z., and Jiang, J. 2016. Using multi-features and ensemble learning method for imbalanced malware classification. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, pp. 965–973.
[24] Guyon, I. and Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, pp. 1157–1182.
[25] Nguyen, T.-T., Huang, J. Z., and Nguyen, T. T. 2015. Unbiased feature selection in learning random forests for high-dimensional data. *The Scientific World Journal* 2015. DOI: 10.1155/2015/471371
[26] Genuer, R., Poggi, J.-M., and Tuleau-Malot, C. 2010. Variable selection using random forests. *Pattern Recognition Letters* 31, 14, pp. 2225–2236.
[27] Al-Azani, S. and El-Alfy, E.-S. 2017. Using word embedding and ensemble learning for highly imbalanced data sentiment analysis in short arabic text. *Procedia Computer Science* 109, pp. 359–366.
[28] McCarthy, R. V., McCarthy, M. M., Ceccucci, W., and Halawi, L. 2019. Predictive Models Using Decision Trees. In *Applying Predictive Analytics*. Springer, Cham, pp. 123–144.