Distance Metric Learning using Particle Swarm Optimization to Improve Static Malware Detection

Martin Jureček and Róbert Lórencz
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic

Keywords: Distance Metric Learning, Malware Detection, Static Analysis, Heterogeneous Distance Function, Particle Swarm Optimization, *k*-Nearest Neighbor.

Abstract: Distance metric learning is concerned with finding appropriate parameters of distance function with respect to a particular task. In this work, we present a malware detection system based on static analysis. We use *k*-nearest neighbors (KNN) classifier with weighted heterogeneous distance function that can handle nominal and numeric features extracted from portable executable file format. Our proposed approach attempts to specify the weights of the features using particle swarm optimization algorithm. The experimental results indicate that KNN with the weighted distance function improves classification accuracy significantly.

1 INTRODUCTION

During the last years, the current trend is to use malware detection frameworks based on machine learning algorithms. Thanks to cloud-based computing which makes the cost of big data computing more affordable, the concept of employing machine learning to malware detection has become more realistic to deploy. The problem to be solved is to detect malware which has never been seen before. While signature-based detection systems (Kephart and Arnold, 1994) identify known malicious programs, these systems can be bypassed by unknown malware. However, the signature-based methods are still popular because of their low false positive rate. Instead of using static signatures, an effective alternative solution is to use machine learning methods to detect malware.

Malware detection techniques can be typically classified into two categories depending on how code is analyzed: static and dynamic analysis. Static analysis (Nath and Mehtre, 2014), (Alrabaee et al., 2016) aims at searching information about structure of a file. Disassembly technique is one of the techniques of static analysis which is used for extracting various features from the executables. Dynamic analysis (Or-Meir et al., 2019), (Egele et al., 2012) aims to examine a program which is executed in a real or virtual environment.

Our research is based on static analysis and feature vectors used in the experiments contains data from the portable executable (PE) file format. Several works (Saad et al., 2019), (Damodaran et al., 2017) have described various limitations of static analysis. The most important drawback is that data captured from static analysis does not describe the complete behavior of a program since the program is not executed. However, dynamic analysis is more time-consuming in comparison to static analysis and there are anti-virtual machine technologies that evade detection systems based on dynamic analysis. Consequently, dynamic analysis could be impractical for a large volume of samples that come to antivirus vendors every day. For these reasons, static analysis has still its place in malware detection systems.

Good similarity measure plays an important role in the performance of geometric-based classifiers, such as *k*-nearest neighbors (KNN). The similarity between two feature vectors is determined by the distance metric between them. The distance between two feature vectors having the same class label must be minimized while the distance between two feature vectors of different classes must be maximized.

A distance metric learning algorithm aims at finding the most appropriate parameters of the metric with respect to some optimization criteria. This task is typically formulated as an optimization problem and in this work, it is related to the malware detection problem. This work concerns with learning a distance function used in the KNN classifier for the malware detection problem. Note that learning the distance metric is an important preprocessing step which is often ignored in practice.

Jureček, M. and Lórencz, R.
Distance Metric Learning using Particle Swarm Optimization to Improve Static Malware Detection.
DOI: 10.5220/0009180807250732
In *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pages 725-732
ISBN: 978-989-758-399-5; ISSN: 2184-4356

The main contribution of this paper is in finding an appropriate weights for the heterogeneous distance function used in the KNN classifier, and as a results, improving classification accuracy of malware detection system. Searching for the most suitable weights with respect to classification accuracy can be considered as an optimization problem. Evolutionary algorithms, swarm algorithms and other heuristics (Luke, 2013) are suitable for our optimization problem. In our experiment, a biologically motivated algorithm called particle swarm optimization (PSO) was used to solve this problem. Experimental results indicate that the performance of KNN using the weighted distance function is considerably better than the performance of KNN without weights.

The rest of the paper is organized as follows. Section 2 briefly reviews some related work in the field of malware detection based on data mining techniques. Some weighted distance functions and distance metric learning techniques are also reviewed in this section. Our proposed malware detection model and theoretical background are presented in Section 3. Experimental setup and results are presented in Section 4. Conclusion and future work are given in Section 5.

2 RELATED WORK

In this section, we briefly review some works related to malware detection based on machine learning techniques. We also review several approaches of how to find the most suitable feature weights for a distance function used in KNN or other techniques working with distances.

2.1 Malware Detection

Over the past two decades, a large number of malware detection techniques has been proposed. To evade malware classifiers, malware writers usually employ obfuscation techniques such as encryption, binary packers, or self-modifying code. In recent years, many malware researchers have focused on data mining and machine learning algorithms to detect unknown malware (Gandotra et al., 2014), (Ye et al., 2017).

(Schultz et al., 2000) were the first who introduced the concept of data mining techniques for detection of malicious code. The authors used three different features: information from the PE header, string features, and byte sequences extracted from binaries. They used three machine learning algorithms: Naive Bayes, Multinomial Naive Bayes, rule induction algorithm called Ripper (Cohen, 1996), and compared them with the signature-based method. Their results indicate that the data mining detection rate of previously unknown malware was twice as high in comparison to the signature-based method.

(Shafiq et al., 2009) extracted structural information from the PE file format and selected the most important features with respect to distinguishing between benign files and malware. The authors have used three feature selection algorithms: Redundant Feature Removal (RFR), Principal Component Analysis (PCA), and Haar Wavelet Transform (HWT) (Witten et al., 2016), and applied five machine learning classifiers: instance based learner (IBk), decision tree (J48), naive bayes (NB), inductive rule learner (RIPPER), and support vector machine (SVM) using sequential minimal optimization. The authors concluded that J48 outperforms the rest of the classifiers in terms of the detection accuracy.

More recently, (Zhong and Gu, 2019) improved performance of deep learning models by organizing them to the tree structure called Multiple-Level Deep Learning System (MLDLS). Each deep learning model focuses on specific malware family. As a result, the MLDLS can handle complex malware data distribution. Experimental results indicate that proposed method outperforms the SVM, decision tree, the single deep learning method and ensemble based approach.

2.2 Weighted Distance Functions for KNN Classifier

The k-nearest neighbors classifier (Cover and Hart, 1967) is one of the simplest and best-known nonparametric algorithms in machine learning. Several approaches have been proposed to increase the performance of KNN. The work (Ghosh, 2006) presents the technique for estimation of the optimal parameter k. Many studies include research on the similarity measures. The work (Yu et al., 2008) studies distance measure based on statistical analysis and presents boosting heterogeneous measure for similarity estimation. Work (Hsu and Chen, 2008) has derived conditions for stability of the distance function in high-dimensional space.

Several distance functions have been presented (Wilson and Martinez, 1997). To improve results, many weighting schemes were proposed. Review of feature weighting methods for lazy learning algorithms was proposed in (Wettschereck et al., 1997).

2.3 Distance Metric Learning

Distance metric learning is an active research area (Yang and Jin, 2006), (Kulis et al., 2013). Distance metric learning is defined as follows. Let $T = \{(x_1, c_1), \ldots, (x_m, c_m)\}$ be the training set of m feature vectors x_i in d-dimensional metric space $\mathcal{S}$, and c_i be class labels. The goal is to learn a linear transformation $L : \mathcal{S} \to \mathcal{S}$, where squared distance between two feature vectors x_i and x_j is defined as $d(x_i, x_j) = \|L(x_i - x_j)\|^2$. Note that d is a valid metric if and only if the matrix L is full rank. We reformulate the definition of the squared distance as $\mathcal{D}(x_i, x_j) = (x_i - x_j)^T M (x_i - x_j)$, where $M = L^T L$. Matrix M is guaranteed to be positive semidefinite and the distance $\mathcal{D}$ is called Mahalanobis metric. Note that when M is equal to the identity matrix, then the distance $\mathcal{D}$ is reduced to Euclidean distance metric. The goal is to find a matrix M which is estimated from the data, that leads to the highest classification accuracy of KNN classifier.

Large Margin Nearest Neighbor (LMNN) (Weinberger et al., 2006) classification is used to learn a Mahanalobis distance metric for KNN classification. LMNN consists of two steps. In the first step, set of k similarly labeled neighbors is identified for each feature vector. In the second step, the Mahalanobis distance metric is learned using convex optimization.

Similar to our approach, (Xu et al., 2017) searched for suitable weight vector using PSO. However, there are several differencies: we used a heterogeneous distance function that can handle both nominal and numeric features, we used different classifier for evaluation, we applied different modification of the PSO algorithm, and our goal is to improve malware detection for a different operating system. (Kong and Yan, 2013) proposed a malware detection method based on structural information. Discriminant distance metric is learned to cluster the malware samples belonging to same malware family.

3 THE PROPOSED MALWARE DETECTION MODEL

In this section, we present our proposed malware detection system and describe all its components. Architecture of the detection system is illustrated in Fig. 1.

The detection system consists of the metric learning phase and the classification phase. First, relevant features are extracted from the binaries. Then we split the dataset into two disjoint subset: T_{pso} for the metric learning phase, and T_{eval} for the classification phase. In the metric learning phase, feature weights are learned from the data. The feature selection method described in Section 3.3 is performed, as a results, dimension of the feature vectors is reduced. Then the data is split into training (80%) and testing (20%) subsets and they are used for computation of the fitness function used in the PSO algorithm.

In the classification phase, we evaluate the best weight vector from the metric learning phase using the KNN classifier. First, we also reduce the dimension of the feature vectors with respect to the feature selection results from the metric learning phase. Then we apply KNN with fivefold cross validation (Picard and Cook, 1984) to obtain reliable experimental results. T_{eval} is randomly divided into five subsets of equal size, where four subsets are used for training and one subset for testing. The experiment is repeated five times on different subsets of data. The accuracies obtained for each fold are averaged to produce a single cross validation estimate.

We do not use cross-validation in the metric learning phase since evaluation of the fitness function is very time consuming. Note that if we used the same training dataset in the metric learning phase and also in the classification phase, we could possibly obtain better classification results than when the training datasets in both phases were disjoint. The aim of this architecture is to show robustness of the resulted feature weights.

3.1 Heterogeneous Distance Metric

In this section, we describe weighted heterogeneous distance function that is used in our experiments. The distance without weights was proposed in (Jureček and Lórencz, 2018). Let $\mathbf{x}$ and $\mathbf{y}$ be two feature vectors of dimension m, and let w_a be weight corresponding to the attribute (feature) a. The weighted distance is defined as follows:

$$\mathcal{D}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{a=1}^{m} w_a^2 d_a^2(x_a, y_a)} \tag{1}$$

where

$$d_a(x,y) = \begin{cases} \mathcal{H}(x,y) & \text{if } a \text{ is a bit array} \\ \delta(x,y) & \text{if } a \text{ is a checksum} \\ \text{Norm\_diff}_a(x,y) & \text{if } a \text{ is a numeric} \\ \text{Norm\_vdm}_a(x,y) & \text{otherwise.} \end{cases} \tag{2}$$

$\mathcal{H}(x,y)$ denotes Hamming distance defined for binary vectors $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n)$ as

$$\mathcal{H}(x,y) = |\{i | x_i \neq y_i, i = 1, \ldots, n\}| \tag{3}$$

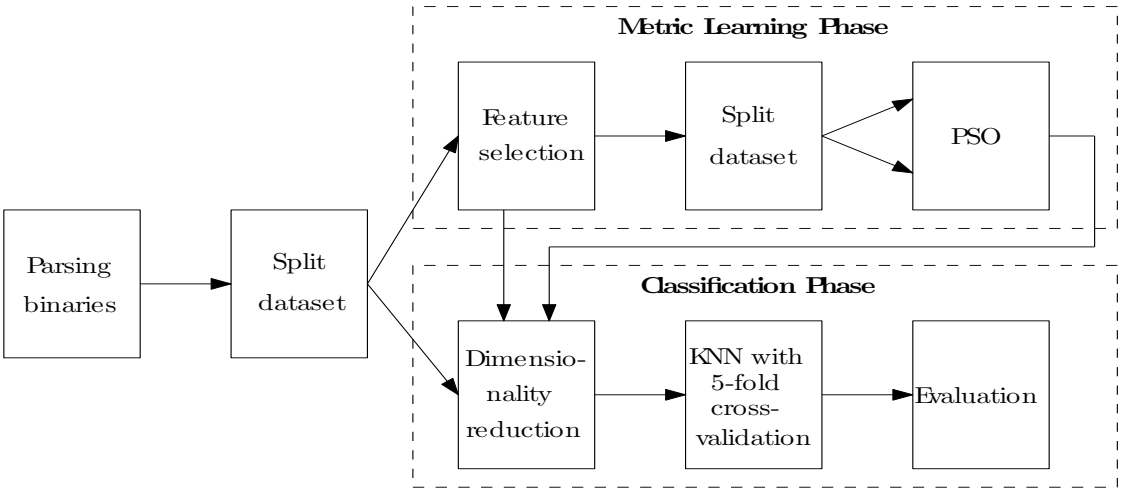

Figure 1: Architecture of our proposed malware detection system.

and $\delta(x,y)$ is the characteristic function defined as

$$\delta(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases} \quad (4)$$

The Value Difference Metric (VDM) was introduced by (Stanfill and Waltz, 1986) and the normalized VDM is defined as

$$\text{Norm\_vdm}_a(x,y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right| \quad (5)$$

where

- C is the number of classes,
- $n_{a,x,c}$ is the number of instances in the training set $\mathcal{T}$ which have value x for attribute a and the instance belongs to class c,
- $n_{a,x}$ is the number of instances in $\mathcal{T}$ that have value x for attribute a.

Function $\text{Norm\_diff}_a(x,y)$ is defined as:

$$\text{Norm\_diff}_a(x,y) = \frac{|x-y|}{4\sigma_a} \quad (6)$$

where σ_a is the standard deviation of the values of numeric attribute a.

The distance $\mathcal{D}$ is a modification of Heterogeneous Value Difference Metric (HVDM) (Wilson and Martinez, 1997), and it can be used for our PE feature space since it handles both numeric and nominal attributes.

3.2 Particle Swarm Optimization

Particle swarm optimization (PSO) is a stochastic optimization algorithm proposed by (Eberhart and Kennedy, 1995). Many variants and modifications of PSO are described in (Wang et al., 2018).

PSO is a biologically motivated algorithm based on swarm intelligence. Each particle is represented as a point in the search space and the quality of each point is determined by a fitness function. Each particle updates its position which is influenced by: the current velocity, previous best particle's position and position of the most successful particle in the swarm.

Concept and notation of the PSO elements with respect to our distance metric learning problem applied on malware detection, is as follows:

- Particle represents vector of weights w. The current position of i-th particle is denoted by x_i and v_i denotes its current velocity.
- Swarm or population is an array of all particles considered in the PSO algorithm.
- Local best position p_i of i-th particle is its best position among all positions visited so far, and $pbest_i$ is the corresponding value of the fitness function f, i.e. $pbest_i = f(p_i)$.
- Global best position p_g is the position of the most successful particle in the swarm, and $gbest_i = f(p_g)$.
- Fitness function f is an objective function that is used to measure the quality of a particle. In our malware detection problem, the fitness function is defined as the accuracy of the KNN classifier.

The pseudocode of the PSO algorithm is presented in Algorithm 1.

Algorithm 1: PSO algorithm.

Input: fitness function f, T_{pso}
Output: vector of weights
1: initialize particles with random positions x_i and velocities v_i
2: **repeat**
3: **for each** particle x_i **do**
4: compute fitness function $f(x_i)$
5: **if** $f(x_i) > pbest_i$ **then**
6: $pbest_i = f(x_i)$
7: $p_i = x_i$
8: **end if**
9: **end for**
10: select the most successful particle in swarm so far, and denote it by p_g
11: **for each** particle x_i **do**
12: $v_i = v_i + \text{Rand}(0,\phi_1) \otimes (p_i - x_i) + \text{Rand}(0,\phi_2) \otimes (p_g - x_i)$
13: $x_i = x_i + v_i$
14: **end for**
15: **until** maximum number of iterations or sufficiently good fitness is attained
16: **return** global best position

$\text{Rand}(0,\varepsilon)$ represents a vector of random numbers uniformly distributed in $[0,\varepsilon]$. Operation $\otimes$ denotes component-wise multiplication. Note the each particle is able to memorize its best previous position and also it knows the best position of the whole swarm so far. Each component of velocity v is kept in the range $[-V_{max}, V_{max}]$, where parameter V_{max} influences search ability of the particles.

To better control the scope of the search and reduce the importance of V_{max}, (Shi and Eberhart, 1998) proposed the following modification of particle's velocity equation (step 12 of Algorithm 1):

$$v_i = \omega v_i + \text{Rand}(0,\phi_1) \otimes (p_i - x_i) + \\ + \text{Rand}(0,\phi_2) \otimes (p_g - x_i), \quad (7)$$

where ω is an inertia weight. Higher values of ω tend to global search while lower values tend to local search. Parameters ϕ_1, ϕ_2 and ω represents the weights and they are used to balance the global and the local search.

The PSO was chosen among other optimization heuristics because its convergence rate is fast and the algorithm is easy to implement and execute in parallel. The drawback of the algorithm is that it is vulnerable to stuck into the local minima.

3.3 Feature Selection

In the proposed approach, the feature vector consists of information from PE file format (Microsoft, 1999) which is the most widely used file format for malware. Gain ratio (GR) (Quinlan, 1986) is used to determine the most useful features with respect to discriminating between malware and benign files.

Gain ratio is a modification of entropy-based measure called information gain (IG) (Mitchell, 1997). Information gain $\text{IG}(\mathcal{T},a)$ is the expected reduction in entropy caused by knowing the value of an attribute a relative to training dataset $\mathcal{T}$, and it is defined as

$$\text{IG}(\mathcal{T},a) = \text{Entropy}(\mathcal{T}) - \sum_{v \in V(a)} \frac{|\mathcal{T}_v|}{|\mathcal{T}|} \text{Entropy}(\mathcal{T}_v), \quad (8)$$

where $V(a)$ denotes the set of all possible values for attribute a, and $\mathcal{T}_v$ denotes the subset of $\mathcal{T}$ for which attribute a has value v.

Gain ratio penalizes attributes with large numbers of possible values by incorporating a term called split information (SI):

$$\text{SI}(\mathcal{T},a) = -\sum_{i=1}^{d} \frac{|\mathcal{T}_i|}{|\mathcal{T}|} \log_2 \frac{|\mathcal{T}_i|}{|\mathcal{T}|}, \quad (9)$$

where $\mathcal{T}_i$ are the d subsets of training dataset $\mathcal{T}$ resulting from partitioning $\mathcal{T}$ by the d-valued attribute a. Split information $\text{SI}(\mathcal{T},a)$ is the entropy of $\mathcal{T}$ with respect to the values of attribute a. The gain ratio is then defined as

$$\text{GR}(\mathcal{T},a) = \frac{\text{IG}(\mathcal{T},a)}{\text{SI}(\mathcal{T},a)}. \quad (10)$$

The more the gain ratio, the more relevant a feature will be.
The following feature set with the highest gain ratio was extracted and used in our experiment:

- Fields from the PE headers: number of sections, date/time stamp, major or minor versions of linker, operating system, image, subsystem; sizes and addresses of data directories; DLL characteristics, and many others.
- Features from sections and their headers: VirtualSize, VirtualAddress, SizeOfRawData, PointerToRawData, Section Flags.
- Resources: number of resources and the number of types of resources.
- Overlay: size of the overlay.
- Other features: entropies and checksums of sections, the size of all imports, the number of DLLs referred, the number of APIs referred.

Detailed description of these features can be found in the documentation (Microsoft, 1999).

3.4 Performance Metrics

In this section, we present the performance metric we used to measure the accuracy of our proposed approach for the detection of unknown malicious codes. For evaluation purposes, the following classical quantities are employed:

- True Positive (TP) represents the number of malicious samples classified as malware
- True Negative (TN) represents the number of benign samples classified as benign
- False Positive (FP) represents the number of benign samples classified as malware
- False Negative (FN) represents the number of malicious samples classified as benign

The performance of our classifier on the test set is measured using three standard parameters. The most intuitive and commonly used evaluation measure in Machine Learning is the accuracy (ACC):

$$\mathrm{ACC} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}} \tag{11}$$

It is defined on a given test set as the percentage of correctly classified instances. The second parameter, True Positive Rate (TPR) (or detection rate), is defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{12}$$

TPR is the percentage of truly malicious samples that were classified as malware. The third parameter is False Positive Rate (FPR), and it is defined as follows:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}} \tag{13}$$

FPR is the percentage of benign samples that were wrongly classified as malware.

4 EXPERIMENTAL SETUP AND RESULTS

In this section, we describe experimental setup and present the results of our experiments.

4.1 Experimental Setup – Dataset and Implementation

In this research, we use dataset consisting of 150,145 Windows programs in the PE file format, out of which 74,978 are malware, and 75,167 are benign programs. The malicious and benign programs were obtained from the laboratory of the industrial partner and also from (VirusShare, 2019).

There are many variants and modifications of the PSO algorithm. Initialization of the population concerns with random generation of particles and their velocities, however there are more advadced methods, such as nonlinear simplex method or centroidal Voronoi tessellations and many others (Wang et al., 2018). In our implementation of modified PSO, results from the feature selection algorithm (described in Section 3.3) are used for initialization of the particles, instead of random initialization. Values of the gain ratio can be considered as particle p and each particle is initialized as $p \otimes \mathrm{Rand}(0,\varepsilon)$, where ε is a small constant. The purpose of this initialization is in the acceleration of PSO, i.e. reducing the searching space is done using results of the feature selection algorithm.

Another modification in our implementation is feature scaling of the weight vector. Since each component w_i of the weight vector has to be non-negative, in computing the fitness function, we use a normalized weight vector where each component is rescaling using min-max normalization:

$$x_{norm} = \frac{x - min}{max - min}, \tag{14}$$

where x is an original value and *min*, resp. *max*, is minimal, resp. maximal value of the original vector.

There are several control techniques (Robinson and Rahmat-Samii, 2004) that are able to avoid particles running out of the search space. In our implementation, positions of particles are not constrained. We use a static topological structure where each particle is fully informed, i.e. it uses information of the entire neighborhood.

Our implementation was executed on a single computer platform having two processors (Intel Xeon Gold 6136, 3.0GHz, 12 cores each), with 32 GB of RAM running the Ubuntu server 18.04 LTS operating system.

4.2 Experimental Results

To ensure a fine tuning of the hyperparameters of our malware detection model, grid search (Bergstra and

Bengio, 2012) was used to explore the following PSO parameters:

- $\phi_1, \phi_2 \in \{0.5, 1., 1.5, 2.\}$,
- $V_{max} \in \{0.5, 1., 2., 4.\}$.

The rest of the PSO parameters are considered as constants: population size is 40, and number of iterations is 30. At the first iteration, inertia weight ω is set to one, and it linearly decreases at each iteration to the value $\omega_{min} = 0.8$. All these parameters were chosen following the guidelines from (Wang et al., 2018) and (Poli et al., 2007). However, the choice of parameters of the PSO is problem-specific, and to determine the parameters appropriately and effectively is an open problem (Wang et al., 2018).

We evaluated the performance of KNN ($k = 5$) with respect to the following initialization techniques in PSO:

- PSO-RAND denotes PSO where position and velocity of each particle are initialized randomly
- PSO-GR denotes PSO where velocity of each particle p is initialized randomly, however, its position is initialized as $p \otimes Rand(0, \varepsilon)$, where ε is a small constant and p is the vector of gain ratio values.

For our experiments, we used the heterogeneous distance function described in Section 3.1. The classification results of the KNN with fivefold cross validation are listed in Table 1.

Table 1: Classification results of the KNN classifier with and without feature weights.

KNN	*TPR*	*FPR*	*ACC*
without weights	96.51%	4.03%	96.24%
PSO-RAND	96.69%	3.46%	96.59%
PSO-GR	96.67%	3.30%	96.72%

Using weighted distance, we reduced the average KNN classification error rate from 3.76% to 3.28%, i.e. the error rate has been decreased by 12.77%.

5 CONCLUSIONS

We applied the PSO algorithm to the problem of finding the most appropriate feature weights used in the heterogeneous distance function defined for the features extracted from PE file format. Our results indicate that classification performance of KNN can be improved by using weighted distance function. By comparing with the experiment of KNN without weights, classification error rate of KNN with weights has been decreased by 12.77%. As a results, the accuracy of malware detection system using a geometric-based classifier such as KNN, can be increased significantly by using appropriate weights of the features.

For future work, it would be interesting to experiment with several distance metric learning algorithms and incorporate them in our proposed malware detection system. Important goal for future work is to learn weights that vary between malware families.

ACKNOWLEDGEMENTS

The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

REFERENCES

Alrabaee, S., Shirani, P., Debbabi, M., and Wang, L. (2016). On the feasibility of malware authorship attribution. In *International Symposium on Foundations and Practice of Security*, pages 256–272. Springer.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Cohen, W. W. (1996). Learning trees and rules with set-valued features. In *AAAI/IAAI, Vol. 1*, pages 709–716.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., and Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1):1–12.

Eberhart, R. and Kennedy, J. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948. Citeseer.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, 5(02):56.

Ghosh, A. K. (2006). On optimum choice of k in nearest neighbor classification. *Computational Statistics & Data Analysis*, 50(11):3113–3123.

Hsu, C.-M. and Chen, M.-S. (2008). On the design and applicability of distance functions in high-dimensional data space. *IEEE Transactions on Knowledge and Data Engineering*, 21(4):523–536.

Jureček, M. and Lórencz, R. (2018). Malware detection using a heterogeneous distance function. *Computing and Informatics*, 37(3):759–780.

Kephart, J. O. and Arnold, W. C. (1994). Automatic extraction of computer virus signatures. In *4th virus bulletin international conference*, pages 178–184.

Kong, D. and Yan, G. (2013). Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1357–1365. ACM.

Kulis, B. et al. (2013). Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu, second edition.

Microsoft (1999). Microsoft portable executable and common object file format specification.

Mitchell, T. M. (1997). Machine learning. *New York*.

Nath, H. V. and Mehtre, B. M. (2014). Static malware analysis using machine learning methods. In *International Conference on Security in Computer Networks and Distributed Systems*, pages 440–450. Springer.

Or-Meir, O., Nissim, N., Elovici, Y., and Rokach, L. (2019). Dynamic malware analysis in the modern era—a state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5):88.

Picard, R. R. and Cook, R. D. (1984). Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization. *Swarm intelligence*, 1(1):33–57.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

Robinson, J. and Rahmat-Samii, Y. (2004). Particle swarm optimization in electromagnetics. *IEEE transactions on antennas and propagation*, 52(2):397–407.

Saad, S., Briguglio, W., and Elmiligi, H. (2019). The curious case of machine learning in malware detection. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, pages 528–535. INSTICC, SciTePress.

Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. (2000). Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 38–49. IEEE.

Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009). Pe-miner: Mining structural information to detect malicious executables in realtime. In *International Workshop on Recent Advances in Intrusion Detection*, pages 121–141. Springer.

Shi, Y. and Eberhart, R. (1998). A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE.

Stanfill, C. and Waltz, D. L. (1986). Toward memory-based reasoning. *Commun. ACM*, 29(12):1213–1228.

VirusShare (2019). Virusshare.com.

Wang, D., Tan, D., and Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft Computing*, 22(2):387–408.

Weinberger, K. Q., Blitzer, J., and Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480.

Wettschereck, D., Aha, D. W., and Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273–314.

Wilson, D. R. and Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of artificial intelligence research*, 6:1–34.

Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Xu, Y., Wu, C., Zheng, K., Wang, X., Niu, X., and Lu, T. (2017). Computing adaptive feature weights with pso to improve android malware detection. *Security and Communication Networks*, 2017.

Yang, L. and Jin, R. (2006). Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2(2):4.

Ye, Y., Li, T., Adjeroh, D., and Iyengar, S. S. (2017). A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, 50(3):41.

Yu, J., Amores, J., Sebe, N., Radeva, P., and Tian, Q. (2008). Distance learning for similarity estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):451–462.

Zhong, W. and Gu, F. (2019). A multi-level deep learning system for malware detection. *Expert Systems with Applications*, 133:151–162.