Received June 2, 2021, accepted June 24, 2021, date of publication July 1, 2021, date of current version July 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094064

Application of Distance Metric Learning to Automated Malware Detection

MARTIN JUREČEK AND RÓBERT LÓRENCZ
Department of Information Security, Faculty of Information Technology, Czech Technical University in Prague, 16000 Prague, Czech Republic
Corresponding author: Martin Jureček (jurecmar@fit.cvut.cz)

This work was supported by the Research Center for Informatics through OP VVV MEYS under Project CZ.02.1.01/0.0/0.0/16 019/0000765.

ABSTRACT Distance metric learning aims to find the most appropriate distance metric parameters to improve similarity-based models such as k-Nearest Neighbors or k-Means. In this paper, we apply distance metric learning to the problem of malware detection. We focus on two tasks: (1) to classify malware and benign files with a minimal error rate, (2) to detect as much malware as possible while maintaining a low false positive rate. We propose a malware detection system using Particle Swarm Optimization that finds the feature weights to optimize the similarity measure. We compare the performance of the approach with three state-of-the-art distance metric learning techniques. We find that metrics trained in this way lead to significant improvements in the k-Nearest Neighbors classification. We conducted and evaluated experiments with more than 150,000 Windows-based malware and benign samples. Features consisted of metadata contained in the headers of executable files in the portable executable file format. Our experimental results show that our malware detection system based on distance metric learning achieves a 1.09 % error rate at 0.74 % false positive rate (FPR) and outperforms all machine learning algorithms considered in the experiment. Considering the second task related to keeping minimal FPR, we achieved a 1.15 % error rate at only 0.13 % FPR.

INDEX TERMS Distance metric learning, malware detection, particle swarm optimization, k-nearest neighbors.

I. INTRODUCTION

The term malware, or malicious software, is defined as any software that does something that causes damage to the user. Malware includes viruses, worms, trojan horses, rootkits, spyware, and any other program that exhibits malicious behavior [1]. In information security, malware attacks have been one of the main threats over the past several decades. While malware developers continuously find new exploitable vulnerabilities, create more and more sophisticated techniques to avoid detection and find new infection vectors, malware analysts and researchers continually improve their defenses. This game seems to have an infinite number of rounds.

The attacker's purpose is no longer to cause damage, such as damaging a computer system without getting money. Nowadays, malware has become a rather profitable business. Malware writers use a variety of techniques to distribute malicious programs and infect devices. They can use self-propagation mechanisms based on various vulnerabilities or use social engineering to trick the user into installing the malware. Malware writers usually employ obfuscation techniques [2] such as encryption, binary packers, or self-modifying code to evade malware classifiers. Many malware researchers have focused on data mining and machine learning (ML) algorithms to defeat these techniques and to detect unknown malware [3]. The performance of many ML algorithms, such as k-Nearest Neighbors (KNN) or k-Means, depends on the distance metric used to measure dissimilarity between samples over some input space. The distance between two samples having the same class label must be minimized while the distance between two samples of different classes must be maximized.

Distance metric learning (DML) aims to automatically learn distance metric parameters from data to improve the performance of classification and clustering algorithms.

The associate editor coordinating the review of this manuscript and approving it for publication was Hao Ji.

Finding the most appropriate parameters of the metric concerning some optimization criterion is typically formulated as an optimization problem. Evolutionary algorithms, swarm algorithms, and other heuristics [4] are suitable for solving this problem. In this work, we used a biologically-motivated algorithm, Particle Swarm Optimization (PSO), to handle this problem related to malware detection.

Most distance metric learning methods learn a Mahalanobis distance with respect to some objective function. The definition of this objective function depends on the training dataset and specific tasks, such as classification or clustering. Malware detection can be defined as a classification problem with two classes: malware and benign samples. The more challenging problem is to cluster malware into malware families [5]. This work empirically demonstrates how to apply distance metric learning to malware detection using a KNN classifier. We experimented with different distance metric learning methods and evaluated them concerning various optimization criteria, such as error rate or its modification.

In this work, we consider portable executables (PE) in the Windows environment. Features consist of metadata from the PE file format [6]. Our proposed detection model is based exclusively on static analysis, aiming to search for information about the file structure without running a program. While the static analysis can be evaded by anti-malware-detection techniques such as obfuscation, it still has a place in the malware detection system since it is much faster than dynamic analysis, which involves running the program.

Malware detection systems that use features originating only from the static analysis can be evaded by obfuscation techniques such as packing [7]. For this reason, we do not suggest using the proposed malware detection system as a standalone and independent application. In order to achieve the highest possible accuracy, it is necessary to use various types of features (byte sequences, API & system calls, opcodes, strings, entropy, instruction traces,…) from both the static and the dynamic analysis. Our work can be used as one component of such a more complex malware detection system.

Note that minimizing the error rate is not the only goal of this work. Another goal is to detect as much malware as possible while maintaining a low false positive rate. From the antivirus vendors' perspective, a false positive error is considered a serious problem. For example, if some legitimate programs integrated into the operating system are detected as malware, then the system could be rendered unusable. False positives can also frustrate developers of the legitimate application that was accidentally blocked by an antimalware system. Since false positives can have serious consequences, we proposed an optimization criterion that takes the cost of false positives into account.

The main contributions of our work are:

Architecture of a malware detection system: We propose the architecture of the malware detection model based on distance metric learning. The detection system processes the data from the PE file format where numeric features are normalized and nominal features are turned into conditional probabilities. Training samples are first used to train the distance metric and then they are used in a KNN classifier with the learned distance metric to classify the testing samples.

Scalable optimization criterion for PSO-based model: To reflect the higher cost of a false positive, we constructed a cost function called *weighted error rate* which we use as a fitness function in the PSO algorithm to minimize error rate and false positive rate.

Application of DML algorithms to malware detection: We explored the use of three state-of-the-art distance metric learning algorithms, namely Large Margin Nearest Neighbor, Neighborhood Component Analysis, and Metric Learning for Kernel Regression, for KNN classification of malware and legitimate software. We compare these models with the PSO-based model and provide practical information concerning performance, computational time and resource usage. We show that the DML-based methods might improve malware classification results even when standard methods such as feature selection or algorithm tuning had already been applied.

The rest of the paper is organized as follows: Section II reviews recent works on malware detection based on machine learning techniques. In Section III, we define the distance metric learning problem and give some theoretical background. Our proposed malware detection model is presented in Section IV. Section V provides an experimental setup. Detailed information about experiments and results is presented in Section VI. Conclusion and future work are given in Section VII.

II. RELATED WORK

The application of machine learning techniques to malware detection has been an active research area for about twenty years. Researchers have tried to apply various well-known techniques such as Neural Networks, Decision Trees, Support Vector Machines (SVM), ensemble methods and many other popular machine learning algorithms. Recent survey papers [8], [9] provide comprehensive information on malware detection techniques using machine learning algorithms.

A. RECENT WORKS

This section briefly reviews some recent works related to malware detection based on machine learning techniques. We mainly focus on works that use the static analysis of Windows PE files, focusing on features extracted from the PE file format.

Wadkar *et al.* [10] proposed a system based on the static features extracted from PE files for detecting evolutionary modifications within malware families. SVM models were trained over a sliding time window, and the differences in SVM weights were quantified using χ^2 statistic. For most of the 13 malware families considered in the experiments, the system detected significant changes.

Yang and Liu [13] proposed a detection model called TuningMalconv with two layers: a raw bytes model in the

TABLE 1. Machine learning-based malware detection systems that are related to this work.

Work	Class	Malware/Benign	Analysis	Features	Dataset
Wadkar [10]	13	26,245/0	static	PE file metadata	Malicia-project [11], VirusShare [12]
Yang [13]	2	15,079/25,986	static	bytes, PE file metadata, string patterns	Malshare [14]
Gao [15]	9	10,868/0	static	bytes, PE file metadata	Kaggle [16]
Xue [17]	63	174,607/0	static & dynamic	bytes, API calls	VXHeavens [18]
Zhong [19]	2	2,242,234/3,425,176	static & dynamic	bytes, API calls	VirusShare [12], Maltrieve [20], VXHeavens [18], Offensive Computing [21], VirusSign [22], private collection
Raff [23]	2	240,000/237,349	static	bytes, PE file metadata	industry partner, VirusShare [12], Open Malware [24]
Kolosnjaji [25]	13	22,694/63	static	PE file metadata, instruction traces	VirusShare [12], Maltrieve [20], private collection
Kumar [26]	2	2,722/2,488	static	PE file metadata	VirusShare [12]
This work	2	74,978/75,167	static	PE file metadata	VirusShare [12], industry partner

first layer and a gradient boosting classifier in the second layer. The feature set was based on static analysis and consisted of raw bytes, n-grams of byte codes, string patterns, and information in the PE header. The experimental results of the TuningMalconv detection model on the dataset with 41,065 samples showed an accuracy of 98.69 %.

Another malware detection model based on static analysis was proposed by Gao *et al.* [15]. The detection model is based on semi-supervised transfer learning and was deployed in the cloud as a SaaS (Software as a Service). The detection model was evaluated on Kaggle malware datasets and improved the classification accuracy from 94.72 % to 96.90 %.

Xue *et al.* [17] proposed a classification system, Malscore, which combines static and dynamic analysis. In static analysis, grayscale images were processed by the Convolutional Neural Network. In dynamic analysis, API call sequences were represented as n-grams and analyzed using five machine learning algorithms: Support Vector Machine, Random Forest, Adaboost, Naïve Bayes, and KNN. The authors performed experiments on more than 170,000 malware samples from 63 malware families and achieved an accuracy of 98.82 %.

Zhong and Gu [19] improved the performance of deep learning models by organizing them in a tree structure called Multiple-Level Deep Learning System. Each deep learning model focuses on a specific malware family. As a result, the Multiple-Level Deep Learning System can handle complex malware data distribution. Experimental results indicate that the proposed method outperforms the Support Vector Machine, Decision Tree, the single Deep Learning method and an ensemble-based approach.

All information on executables used in the work proposed by Raff *et al.* [23] came from the PE header, more specifically, from the MS-DOS, the COFF (Common Object File Format), and the Optional Header. Neural networks were trained from raw bytes which were not parsed for explicit features, and as a result, no preprocessing or feature engineering was required. More than 400,000 samples were used for training, and the Fully Connected Neural Network model achieved the highest accuracy.

Kumar *et al.* [26] proposed a malware detection system which uses machine learning techniques and is based exclusively on static analysis. The dataset contained 2,722 malware and 2,488 benign program samples, and the original feature set consisted of 53 PE file header fields from the DOS header, File header, and Optional header. These features were then processed, and 68 integrated features were derived. In the experiments, six machine learning algorithms Logistic Regression, Linear Discriminant Analysis, Random Forest, k-Nearest Neighbors, Decision Tree, and Gaussian Naïve Bayes were used. The highest classification accuracy, 98.4 %, was achieved by Random Forest.

Kolosnjaji *et al.* [25] proposed a neural network architecture that combines convolutional and feed-forward neural layers. The authors used only the static malware analysis where inputs to feed-forward layers were the fields of the PE header while inputs to the convolutional layers were assembly opcode sequences. The proposed hybrid neural network achieved 93 % on precision and recall.

Table 1 summarizes related works and our work in terms of the number of classes, the size of the dataset, the type of analysis, features used, and the source of the dataset.

B. DISTANCE METRIC LEARNING-BASED WORKS

Surprisingly, there is a distinct lack of experimentation with distance metric learning techniques applied on large and real-world datasets from the Windows environment. In the rest of the section, we briefly mention two of our previous works on malware detection methods that rely on distance metric learning. This paper can be considered as an extension of them. In [27], we applied the Particle Swarm Optimization algorithm to the problem of finding the appropriate feature weights used in the heterogeneous distance function [28] specifically defined for the PE file format to classify malware and benign files. We showed that the error rate of the KNN classifier could be decreased by 12.77 % using the weighted distance function. Our other work [5] focused on the application of three distance metric learning methods applied to the multiclass classification problem with seven classes:

six prevalent malware families and the benign files. Using Metric Learning for Kernel Regression method to learn the Mahalanobis distance metric, we achieved average precision and recall of 97.04 %, both using the KNN classifier.

III. PROBLEM STATEMENT AND BACKGROUND

This section provides basic information on distance metric learning and briefly discusses three selected distance metric learning methods used in our experiments.

Euclidean distance is by far the most commonly used distance. Let $\mathbf{x}$ and $\mathbf{y}$ be two feature vectors from real $n-$dimensional space $\mathbb{R}^n$, and let $w_i, i = 1, \ldots, n$, be a non-negative real number associated with the i-th feature. The weighted Euclidean distance is defined as:

$$d_w(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i^2 (x_i - y_i)^2} \tag{1}$$

The goal of learning the weighted Euclidean distance is to find the best weight vector $w = (w_1, \ldots, w_n)$ with respect to some optimization criterion, usually the minimal error rate. Several other distance functions have been presented [29]. In order to improve results, many weighting schemes were proposed. A review of feature weighting methods for lazy learning algorithms was proposed in [30].

Mahalanobis distance is another popular distance. It is defined for two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ of dimension n as

$$d_M(\mathbf{x}, \mathbf{y}) = \sqrt{(x - y)^\top M (x - y)}, \tag{2}$$

where M is a positive semidefinite matrix. Mahalanobis distance can be considered as a generalization of Euclidean distance since the Euclidean distance can be expressed as a Mahalanobis distance where M is the identity matrix. If M is diagonal, then this corresponds to learning the feature weights $M_{ii} = w_i$ defined for weighted Euclidean distance in Eq. (1).

The goal of learning the Mahalanobis distance is to find the best matrix M with respect to some optimization criterion. Regarding the KNN classifier employed in this work, the main goal is to find a matrix M which is estimated from the training set that leads to the lowest error rate of the KNN classifier. Another goal of this work is to minimize the error rate taking into account the cost of false positives. Since a positive semidefinite matrix M can always be decomposed as $M = L^\top L$, the distance metric learning problem can be viewed as finding either M or $L = M^{\frac{1}{2}}$. Therefore, Mahalanobis distance defined in Eq. (2) can be expressed in terms of the matrix L as

$$d_M(\mathbf{x}, \mathbf{y}) = d_L(\mathbf{x}, \mathbf{y}) = \|L^\top (x - y)\|_2 \tag{3}$$

Another application of distance metric learning is dimensionality reduction. The matrix $\mathbf{L}$ can be used to project the original feature space into a new embedding feature space. This projection is a linear transformation defined for feature vector $\mathbf{x}$ as

$$\mathbf{x}' = \mathbf{L}\mathbf{x} \tag{4}$$

Mahalanobis distance of two points x, y from the original space defined in Eq. (2) corresponds to the Euclidean distance between transformed points $x' = Lx, y' = Ly$ defined as follows:

$$d_L(x, y) = \|L^\top (x - y)\|_2 = \sqrt{(x' - y')^\top (x' - y')} \tag{5}$$

This transformation is useful since the computation of Euclidean distance has lower time complexity than that of the Mahalanobis distance.

Distance metric learning has attracted a lot of attention in the machine learning field and is still an active research area [31]. There have been many proposed methods [32]–[34]. Next, we briefly describe three state-of-the-art distance metric learning methods that we used in our experiments. Specifically, the weighted Euclidean distance was learned by the Particle Swarm Optimization algorithm, and the Mahalanobis distance was learned by the distance metric learning methods described in the rest of this section.

A. LARGE MARGIN NEAREST NEIGHBOR

Large Margin Nearest Neighbor (LMNN) [35] is one of the state-of-the-art distance metric learning algorithms used to learn a Mahalanobis distance metric for a KNN classification. LMNN consists of two steps. In the first step, for each instance x a set of k nearest instances belonging to the same class as x (referred as *target neighbors*) is identified. In the second step, we adapt the Mahalanobis distance to reach the goal that the target neighbors are closer to x than instances from different classes separated by a large margin. The Mahalanobis distance metric is estimated by solving the semidefinite programming problem defined as:

$$\min_{\mathbf{L}} \sum_{i,j:j \rightsquigarrow i} \Big(d_L(x_i, x_j)^2 + \mu \sum_{k:y_i \neq y_k} [1 + d_L(x_i, x_j)^2 - d_L(x_i, x_k)^2]_+ \Big) \tag{6}$$

The notation $j \rightsquigarrow i$ refers that the sample $\mathbf{x_j}$ is a *target neighbor* of the sample $\mathbf{x_i}$, and y_i denotes the class of $\mathbf{x_i}$. The parameter μ defines a trade-off between the two objectives:

1) to minimize the distances between samples $\mathbf{x_i}$ and their target neghbors $\mathbf{x_j}$,
2) to maximize the distances between samples $\mathbf{x_i}$ and their impostors $\mathbf{x_k}$ which are samples which belong among the nearest neighbors of $\mathbf{x_i}$ but have different class labels (i.e. $y_i \neq y_k$).

Finally, $[x]_+$ is defined as the hinge-loss, i.e. $[x]_+ = \max\{0, x\}$. In [36], LMNN was extended to multiple local metrics and the learning time of LMNN was reduced using metric ball trees.

B. NEIGHBORHOOD COMPONENT ANALYSIS

Goldberger *et al.* [37] proposed the Neighborhood Component Analysis (NCA), which is a distance metric learning algorithm specially designed to improve the KNN classification.

Let p_{ij} be the probability that the sample $\mathbf{x_i}$ is the neighbor of the sample x_j belonging to the same class as $\mathbf{x_i}$. This probability is defined as:

$$p_{ij} = \frac{\exp(-||\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_j||_2^2)}{\sum_{l \neq i} \exp(-||\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_l||_2^2)}, \quad p_{ii} = 0 \tag{7}$$

The goal of NCA is to find the matrix $\mathbf{L}$ that maximizes the sum of probabilities p_i:

$$\arg\max_L \sum_i \sum_{j:j \neq i, y_j = y_i} p_{ij} \tag{8}$$

The gradient ascent algorithm solves this optimization problem. Neither LMNN nor NCA algorithms make any assumptions on the class distributions.

C. METRIC LEARNING FOR KERNEL REGRESSION

Weinberger *et al.* proposed Metric Learning for Kernel Regression (MLKR) [38] which aims at training the Mahalanobis matrix by minimizing the error loss over the training samples:

$$\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2, \tag{9}$$

where the prediction class $\hat{y}_i$ is derived from kernel regression by calculating a weighted average of the training samples:

$$\hat{y}_i = \frac{\sum_{j \neq i} y_j K(x_i, x_j)}{\sum_{j \neq i} K(x_i, x_j)} \tag{10}$$

MLKR can be applied to many types of kernel functions $K(x_i, x_j)$ and distance metrics $d(x_i, x_j)$.

Recall that the mentioned distance metric learning algorithms can be used as supervised dimensionality reduction algorithms. Considering the matrix $L \in \mathbb{R}^{d \times n}$ with $d < n$ then the dimension of transformed sample $x' = Lx$ is reduced to d.

IV. PROPOSED MODEL

In this section, we describe our proposed malware detection model based on distance metric learning. First, the features description and engineering are provided. Then we describe the modification of the Particle Swarm Optimization algorithm, which we used to find appropriate feature weights of weighted Euclidean distance. Finally, we complete this section by proposing the architecture of the malware detection model.

A. FEATURE DESCRIPTION

The features used in our experiments are extracted from the PE file format [6] which is the file format used for executables, DLLs, object code and other files used in the 32 and 64-bit versions of the Windows operating system. The PE file format is the most widely used file format for malware samples that run on desktop platforms. Before describing the features used in our experiments, let us first examine the short outline of the PE file format.

A PE file consists of headers and sections that encapsulate the information necessary to manage the executable code. The PE file header provides all the descriptive information concerning the locations and sizes of structures in the PE file to the loader process. The header of a PE file consists of the DOS header, the PE signature, the COFF file header, the optional header and the section headers. The optional file header is immediately followed by the section headers which provide information about sections, including their locations, sizes, and characteristics.

Sections divide the file content into code, resources and various types of data. The order of the sections is not the same for each PE file. Moreover, malware authors can change the names of the sections. Therefore, we prefer to consider only the order of sections rather than the name of the sections (such as.text,.data,.rsrc). The last section of a PE file may be of particular importance. It may contain useful information, especially for some types of malware, e.g., the file infector which typically attaches malicious code at the end of the file. To deal with a various number of sections across the samples, we have decided to consider only the first four sections and the last section.

Based on our empirical studies and the PE format analysis, we selected a set of static features that help to distinguish malware and benign files. The features used in our experiments are of three types: nominal, numeric, and bit fields. In the following section, we describe how these three types of features were preprocessed.

Let $T = \{(x_1, c_1), \ldots, (x_m, c_m)\}$ be the training set, where x_i is a feature vector and $c_i = cl(x_i)$ is the corresponding class label. In our binary classification task, we will consider two classes $\mathcal{C}$ and $\mathcal{M}$, where $\mathcal{C}$ denotes the class of benign samples and $\mathcal{M}$ denotes the class of malware. Let each sample be represented by the feature set $\{f_1, \ldots, f_n\}$. Let the feature f_j be nominal and let s be the feature vector corresponding to some unknown sample. Then $P(cl(s) = \mathcal{M}|f_j = h)$ denotes the conditional probability that the output class of s is malware given that feature f_j has the value h. Using data from training set T, we estimate this probability as

$$P(cl(s) = \mathcal{M}|f_j = h) = \frac{n_{f_j,h,\mathcal{M}}}{n_{f_j,h}}, \tag{11}$$

where

- $n_{f,x,c}$ is the number of samples in the training set T which have value x for feature f and the sample belongs to class c,
- $n_{f,x}$ is the number of samples in T that have value x for feature f.

If some feature vector s from the testing set would have previously unseen value h of some feature f_j then we set

$$P(cl(s) = \mathcal{M}|f_j = h) = P(cl(s) = \mathcal{M}) \approx 1/2$$

with respect to our dataset.

Following this approach, for each sample s, we transform each value h of each nominal attribute f_j according to the following rule:

$$h \longmapsto P(cl(s) = \mathcal{M} | f_j = h) \tag{12}$$

Regarding numeric features, it is necessary to take into account their different ranges. Therefore the following data normalization method is employed on each numeric feature f to rescale its original value h using *min-max normalization*. For each feature vector s, we transform each value h of each numeric feature f according to the following rule:

$$h_{norm} = \frac{h - f_{min}}{f_{max} - f_{min}}, \tag{13}$$

where f_{min}, resp. f_{max}, is the minimal, resp. the maximal value among all known values of the feature f.

To handle features that are bit arrays $(b_1, \ldots, b_k)$, we split up each component b_i from the array and consider it as an independent feature. Finally, after preprocessing all three types of features, we apply several feature selection and extraction algorithms and select the most relevant features.

B. FINDING THE FEATURE WEIGHTS USING PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) [39] is a biologically-motivated stochastic optimization algorithm based on swarm intelligence. Each particle is represented as a point in the search space, and a fitness function determines the quality of each point. Each particle updates its position, which is influenced by the current velocity, the previous best particle's position, and the most successful particle in the swarm.

The concept and notation of the PSO elements for finding the feature weights used in the weighted Euclidean distance Eq. (1) in the KNN classification is as follows:

- A particle is represented as a vector of weights w. The current position of i-th particle is denoted by x_i and v_i denotes its current velocity.
- A swarm or population is an array of all particles considered in the PSO algorithm.
- The 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. $pbest_i = f(p_i)$.
- The global best position p_g is the position of the most successful particle in the swarm, and $gbest_i = f(p_g)$.
- The fitness function f is an objective function used to measure the quality of a particle. In our malware detection problem, the optimization criterion can be defined as the error rate of the KNN classifier. In this work, we will also consider another optimization criterion focused on minimizing the false positive rate.

The PSO algorithm has three inputs: the fitness function f, a training set T_{pso}, and vector p of feature importance scores [40] obtained by the feature selection algorithm. The pseudocode of the modified PSO algorithm is presented in Algorithm 1.

Rand$(0, \epsilon)$ represents a vector of random numbers uniformly distributed in $[0, \epsilon]$ where ϵ is a small constant.

Algorithm 1 PSO Algorithm

```
Input: fitness function $f$, $T_{pso}$, $p$
Output: vector of weights
 1: initialize particles:
        $x_i = p \otimes Rand(0, \epsilon_1)$
        $v_i = Rand(-\epsilon_2, \epsilon_2)$
 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:         $v_i = \omega v_i + \text{Rand}(0, \phi_1) \otimes (p_i - x_i) + \text{Rand}(0, \phi_2) \otimes$
            $(p_g - x_i)$
13:         $x_i = x_i + v_i$
14:     end for
15: until maximum number of iterations is attained
16: return global best position
```

Operation $\otimes$ denotes a component-wise multiplication. Note that each particle can memorize its best previous position, and it also 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 the parameter V_{max} influences search ability of the particles. An inertia weight ω is used to better control the search scope and reduce the importance of V_{max}. Higher values of ω tend to prefer the global search while lower values tend to prefer the local search. Parameters ϕ_1 and ϕ_2 represent the weights and are used to balance the global and the local search. The purpose of the initialization is in the acceleration of PSO, i.e., reducing the searching space is done using the feature selection algorithm results.

This work concerns the classification problem where the definition of the fitness function depends on the KNN classifier. The fitness function of the clustering problem can alternatively be defined using purity or silhouette coefficient.

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 getting stuck in the local minima.

In the rest of this section, we propose the optimization criteria for detecting as much malware as possible while keeping a low false positive rate. To consider the different costs of a false positive and false negative, we adjust the loss function that penalizes false positives.

Since our dataset is well-balanced, we consider the error rate as the appropriate measure of performance. The error rate is defined as the percentage of incorrectly classified instances. We can rewrite the error rate in terms of the number

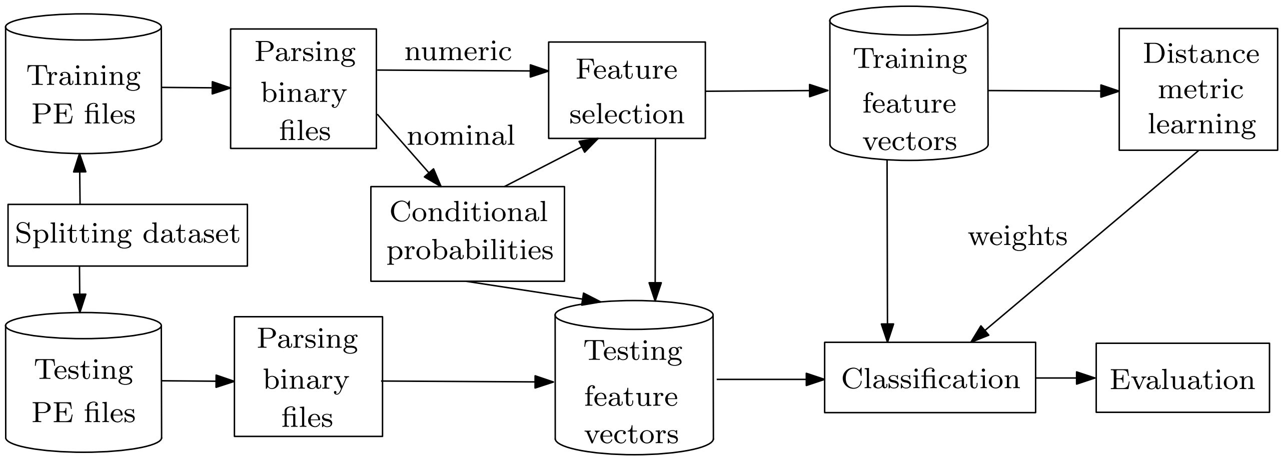


FIGURE 1. **Architecture of our proposed malware detection model using distance metric learning.**

of false positives (FP) and number of false negatives (FN) as

$$\mathrm{ERR} = \frac{\mathrm{FP} + \mathrm{FN}}{|T_{test}|}, \tag{14}$$

where $|T_{test}|$ is the number of testing samples. We modify Eq. (14) by adding the parameter $c > 1$ which corresponds to the cost for false positive. Then we define the optimization criterion called *weighted error rate* (WERR), which takes into account the cost of the false positive:

$$\mathrm{WERR} = \frac{c\,\mathrm{FP} + \mathrm{FN}}{|T| + (c-1)\mathrm{FP}} \tag{15}$$

One interpretation of WERR is that if we would change the parameters of the classifier and achieve the same error rate (with possibly different FP_{new} and FN_{new} compared to the original values of FP_{old} and FN_{old}) then these results will be better with respect to WERR if

$$c\,\mathrm{FP}_{new} - \mathrm{FP}_{old} < \mathrm{FN}_{old} - \mathrm{FN}_{new} \tag{16}$$

One aspect of the WERR criterion is that we can "exchange" one false positive for c false negatives while keeping the error rate unchanged. Note that when $c = 1$, then WERR is equal to the error rate. In all experiments, we used the WERR criterion as a fitness function of the PSO algorithm. When not mentioned, the value of the parameter c was set to one.

C. ARCHITECTURE

We present the malware detection system based on distance metric learning. The system uses static analysis of PE file headers and sections. The proposed architecture is depicted in Figure 1 and outlined in the following seven basic steps:

Step 1: **Splitting the data.** The set of samples is randomly divided into the training set and the testing set. The training set is used for training a distance metric (see step 5) and a classifier (see step 6). The testing set is used for testing the classifier with the learned distance metric.

Step 2: **Parsing binaries.** For each sample from the training and the testing set, we extract and store information from the PE file format. We use Python module `pefile` [41] for extracting the features. These features will be preprocessed in the step 3. In step 4, only the relevant features will be selected and considered in experiments.

Step 3: **Preprocessing of features.** Conditional probability $P(x \text{ is malware}|x_i = h)$ is computed for each nominal feature x_i and for each value h of the feature x_i that appears in training set. Numeric features are normalized according to *min-max normalization*. Bit arrays are split up into single boolean features.

Step 4: **Feature selection.** The feature selection algorithm is used to determine the relevant features and produce the final version of the feature set.

Step 5: **Learning the distance metric.** The distance metric learning method is applied to the training feature vectors in order to produce the appropriate distance metric parameters. In the case of high computational complexity, only a subset of training vectors can be used to learn the distance metric.

Steps 6: **Classification.** The distance metric learned in step 5 is used in the KNN classifier to classify samples from the testing set.

Steps 7: **Evaluation:** Performance metrics, such as true positive, false positive and error rate, are used to measure the classification results.

The computation of the conditional probabilities for nominal features and the execution of the feature selection algorithm for all three types of features is only performed on the training samples. The corresponding conditional probabilities and selected features are applied to design both the training and testing feature vectors.

V. EXPERIMENTAL SETUP

In this section, we present a detailed description of the experimental setup. First, we introduce the dataset used in our experiments. Then, we describe performance metrics and present the results of feature selection.

A. DATASET AND IMPLEMENTATION

We validated our approach using datasets containing real-world data from 150,145 Windows programs in the PE file format, out of which 74,978 were malicious, and 75,167 were benign. The malicious and benign programs were obtained from the industrial partner's laboratory and from the Virusshare repository [12]. Our dataset contains both obfuscated (e.g., packed and/or polymorphic) and non-obfuscated samples. We confirm that all malicious samples considered in our experiments match known signatures from anti-virus companies. Also, none of our benign programs were detected as malware.

We used Python module `pefile` [41] for extracting features from the PE files. This module extracts all PE file attributes into an object making them easily accessible. We extracted 370 features based on static information only, i.e., without running the program. The dimensionality is high since in each section each flag of each kind of characteristic was considered as a single feature.

Our implementations of the feature selection algorithms, DML algorithms, the ML classifiers and the classification metrics are based on the Scikit-learn library [42]. If not specifically mentioned, the hyperparameters of the ML classifiers and the DML methods were set to their default values in the Scikit-learn library.

Our implementation was executed on a single computer with two processors (Intel Xeon Gold 6136, 3.0GHz, 12 cores each), with 64 GB of RAM running the Ubuntu server 18.04 LTS operating system. Memory usage was not exceeded in any experiment conducted in this work.

B. PERFORMANCE METRICS

This section presents the performance metrics we used to measure the accuracy of our proposed approach. 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 metrics. The most intuitive and commonly used evaluation metric in machine learning is the error rate (ERR):

$$\mathrm{ERR} = \frac{\mathrm{FP} + \mathrm{FN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{17}$$

It is defined on a given test set as the percentage of incorrectly classified instances. An alternative for ERR is accuracy defined as ACC $=$ 1$-$ERR. The second parameter, True Positive Rate (TPR) (or detection rate), is defined as:

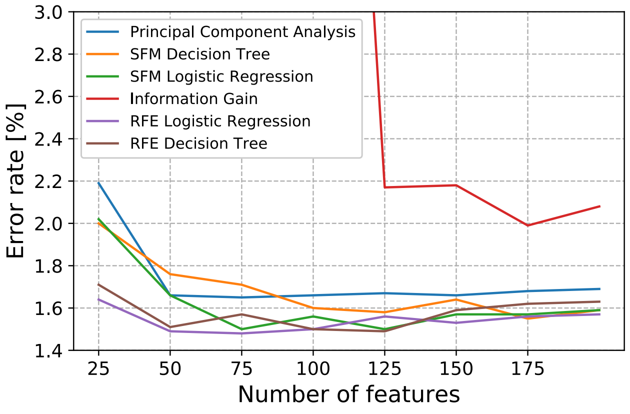


FIGURE 2. Evaluation of the feature selection algorithms in terms of error rates of the KNN ($k = 3$) classifier. The abbreviation SFM refers to procedure `feature_selection.SelectFromModel`, and the abbreviation RFE refers to Recursive Feature Elimination implemented in `feature_selection.RFE`, both from the Scikit-learn library.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{18}$$

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:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{19}$$

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

C. FEATURE SELECTION ALGORITHMS

To reduce the high dimension of the feature vector we used a feature selection algorithm to select the most relevant subset of features. We applied six feature selection algorithms and evaluated them using the KNN ($k = 3$) classifier. Figure 2 shows that the highest accuracy was achieved with the Recursive Feature Elimination (RFE) Logistic Regression for 75 selected features. The feature selection algorithms were evaluated on the whole training data, that is, 70% of samples of all 150,145 samples. To make our results reproducible, Table 7 in Appendix A summarizes the 75 selected features used in our experiments. We kept the name of the fields in the same form as in the documentation [43] so that the reader can easily find detailed description. In all following experiments, we used the dataset processed by the RFE logistic regression, which reduced the dimensionality from 370 to 75.

VI. EXPERIMENTAL RESULTS

A collection of experiments concerning distance metric learning techniques has been conducted. Firstly, we compared the DML techniques and performed additional experiments with the two most successful techniques. Then we focused on minimizing the false positive rate, and finally we compared our approach based on PSO to the state-of-the-art machine learning algorithms.

We first searched for the hyper-parameters of the DML methods. Appropriate hyper-parameters can have large impact on the predictive or computation performance. Tuning the hyper-parameters of the LMNN algorithm using grid search exhaustively considers all parameter combinations. The following searching grids were explored: Number of nearest neigbors $k \in \{1, 3, 5, 7, \ldots, 21\}$, Maximum number of iterations of the optimization procedure $n_{max} \in \{500, 1000, 1500, 2000, 2500, 3000\}$, learning rate of the optimization procedure $r \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$. Note that in all experiments with LMNN we used the parameter $\mu = 1$ defined in Eq. (6) as the trade-off between the two objectives. The lowest error rate was achieved with the following LMNN hyper-parameters: number of neighbors $k = 3$, maximum number of iterations $n_{max} = 1000$ and learning rate $r = 10^{-6}$. These hyper-parameters were used in all successive experiments. We left the hyper-parameters of NCA and MLKR at the default values provided by in the Scikit-learn library. Regarding the PSO algorithm, we explored the following PSO parameters: $\phi_1, \phi_2 \in \{0.5, 1, 1.5, 2\}$, and $V_{max} \in \{0.5, 1, 2, 4\}$. The lowest error rate was achieved with the following hyper-parameters: $\phi_1 = \phi_2 = 1$, and $V_{max} = 2$. The rest of the PSO parameters were as follows: population size is 40 and number of iterations is 30. For 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 of these PSO parameters were chosen according to guidelines from [44].

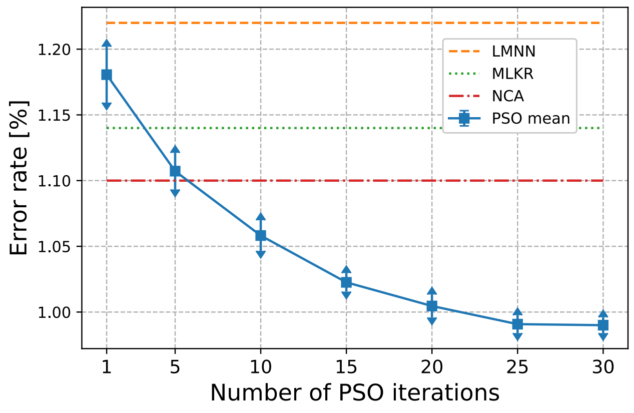


FIGURE 3. Average classification error with the standard deviation is illustrated as a function of the maximum number of iteration in the PSO algorithm.

We ran the PSO algorithm ten times. Figure 3 illustrates the mean and standard deviation of the error rate corresponding to the various number of iterations. The PSO algorithm was run with 50 iterations, however, in each run the algorithm converged to the local minima before reaching 30 iterations. Note that even after the first iteration, PSO outperforms LMNN. The reason lies in the initialization step of the Algorithm 1. Positions of particles in the initialization step of PSO were set according to the feature importance score computed in the feature selection step rather than randomly. As a result, PSO was accelerated and better classification results were achieved.

TABLE 2. The performance of three selected distance metric learning algorithms compared with the performance of the PSO-based model and the non-learned model referred to Euclidean.

Method	TPR[%]	FPR[%]	ERR[%]	Learning time
Euclidean	97.66	0.31	1.33	–
LMNN	97.88	0.32	1.22	2h 7min
MLKR	98.04	0.32	1.14	80h 11min
NCA	98.08	0.28	1.10	10h 20min
PSO	**98.25**	**0.22**	**0.99**	**1h 5min**

A. COMPARISON OF DISTANCE METRIC LEARNING ALGORITHMS

Several distance metric learning algorithms such as LMNN, NCA, and MLKR were designed to improve the KNN classifier. For this reason, these three algorithms were included in our experiments. Table 2 shows the performance of the KNN classifier ($k = 3$) using the common Euclidean distance, the Mahalanobis distance learned by three selected DML algorithms, and the weighted Euclidean distance learned by the PSO algorithm. The KNN classifier achieved the lowest error rate for the weighted Euclidean metric learned by the PSO algorithm.

Recall that while PSO aims at learning a diagonal matrix, the goal of LMNN, NCA, and MLKR is to learn a full matrix. Due to the high computational complexity of the DML algorithms, we conducted the experiment for the randomly chosen subset of the training dataset. Distance metric learning algorithms were trained on 50,000 samples, and the KNN classifier with learned distance was tested on 21,430 samples. These numbers of samples follow the ratio of 70:30 in the sizes of training and testing sets. Based on the trade-off between minimizing the error rate and execution time, we chose only LMNN and PSO for the rest of the experiments.

B. ADDITIONAL EXPERIMENTS FOR LMNN AND PSO

1) COMPARISON OF LMNN AND PSO

In the first experiment, we explored the performance of the LMNN-based model and the PSO-based model for different sizes of datasets. The experiment was conducted ten times for randomly chosen training and testing datasets keeping the 70:30 ratio of their sizes. The number of training samples, the learning method, and the average learning times, TPR, FPR, and ERR estimated on the testing set are summarized in Table 3. The number of nearest neighbors considered in both LMNN-based and PSO-based models was $k = 3$.

For smaller datasets (i.e., a few thousand samples), the LMNN-based model achieved a lower error rate with approximately the same learning time as the PSO-based model. The results indicate that with the increasing volume of data the ratio of computing time and error rate decreases in favor of PSO.

2) THE EFFECT OF PARAMETER *k*

We discuss how different parameter settings of k (i.e., number of neighbors) affect the performance of the KNN classifier. We explored the variation of error rates for the following

TABLE 3. Comparison of the LMNN-based model and the PSO-based model for various sizes of datasets.

# training samples	Learning method	Learning time	TPR[%]	FPR[%]	ERR[%]
1000	LMNN	8s	96.26	4.17	3.95
	PSO	9s	95.92	1.44	2.79
5000	LMNN	2min 34s	96.51	1.90	2.71
	PSO	2min 32s	97.41	1.97	2.29
20000	LMNN	23min 51s	97.94	1.50	1.78
	PSO	18min 11s	98.39	1.49	1.55
50000	LMNN	2h 6min 40s	97.88	0.32	1.22
	PSO	1h 5min 20s	98.25	0.22	0.99

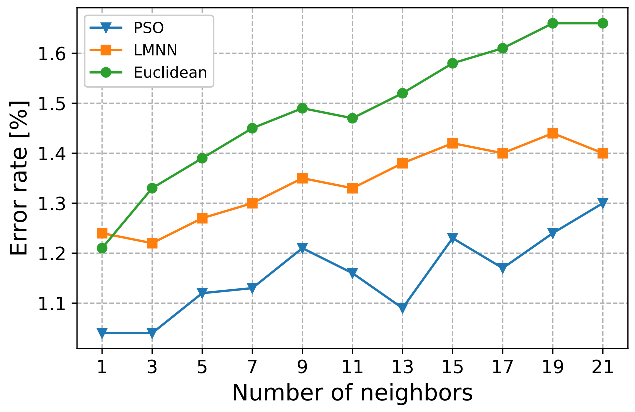


FIGURE 4. The relation between the number of neighbors and the performance of KNN using various distances.

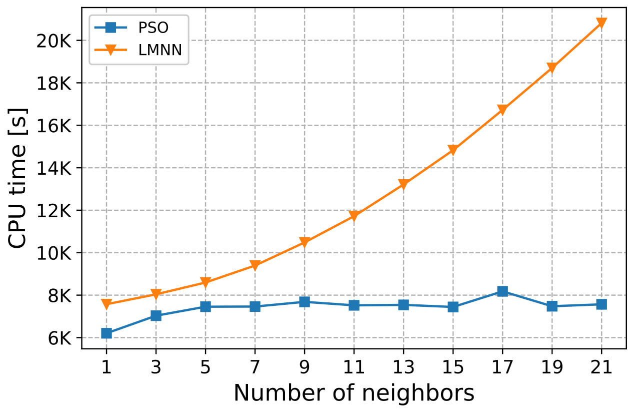


FIGURE 5. Learning time of LMNN and PSO (with 30 iterations). The experiment was conducted ten times, and the computational times were averaged.

three variants: Euclidean distance (without feature weights), Mahalanobis distance learned by LMNN, and weighted Euclidean distance where the weights were computing using PSO. For these three variants, the KNN was trained on 50,000 training samples, and the error rates were estimated on 21,430 testing samples. The experiment was performed ten times, and Figure 4 shows the averaged results of the KNN classifier for various values of the parameter k from the set $\{1, 3, 5, \dots, 21\}$.

In the additional experiments, we explored the relation between the number of neighbors and the learning time. Figure 5 shows that with the increasing number of neighbors the learning time of PSO increases only negligibly compared to the learning time of LMNN.

3) LMNN PROJECTION OF THE ORIGINAL FEATURE SPACE

In the next experiment, we used the weight matrix L, defined in Eq. (3) and learned by LMNN, to project the original feature space into a new embedding feature space. We followed the goal that the k nearest neighbors of each instance belong to the same class while a large margin separates instances from different classes. Recall that this projection is a linear transformation defined as $x' = Lx$. This experiment aims to illustrate the difference between the original (non-transformed) data and the LMNN-transformed data. Two-dimensional embedding of 700 samples using the t-SNE algorithm [45] is shown in Figure 6 where similarity plots for four scenarios are compared.

4) LIMITATIONS OF LMNN FOR LARGER DATASETS

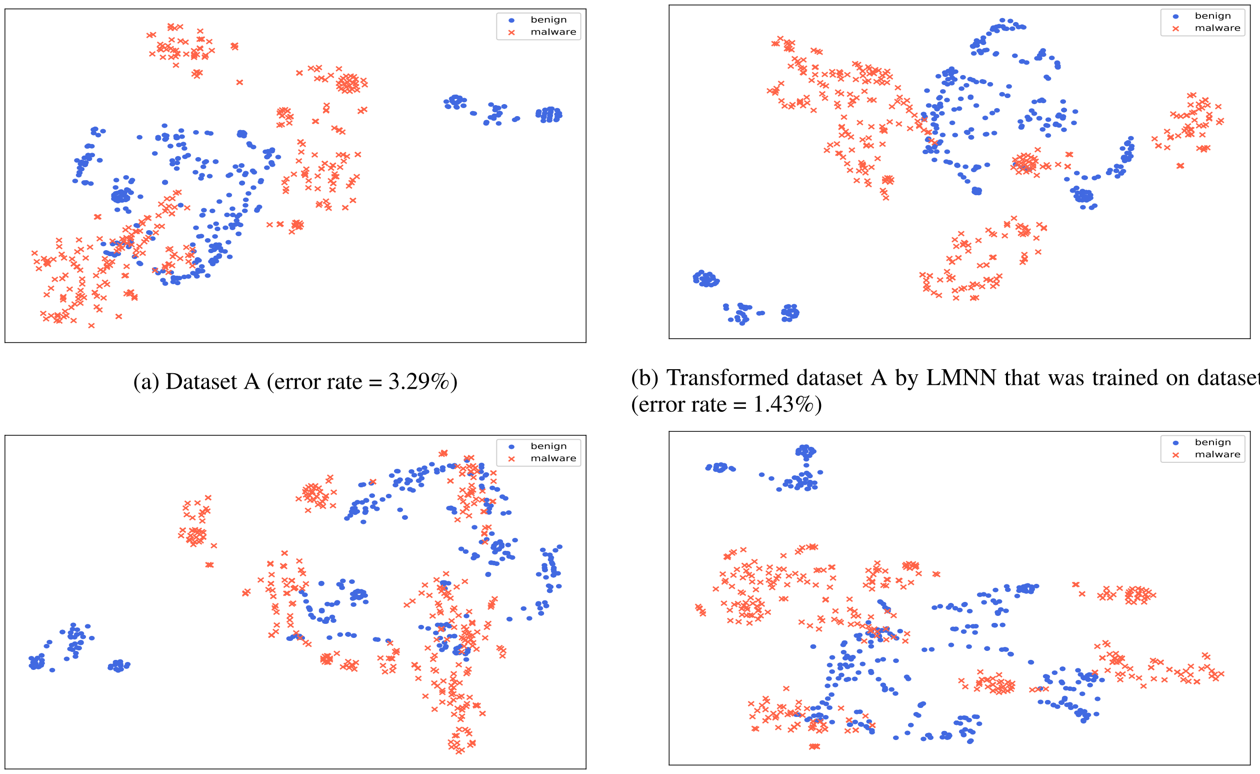
The result of the DML algorithms for the Mahalanobis distance metric is a $n \times n$ matrix where n is the dimension of the feature vector. Since the number of components of the matrix grows at a quadratic rate with n and the size of the training data is fixed, we can expect that the size of training data stops being sufficient for high values of n. Note that the PSO-based model using the weighted Euclidean distance needs only n parameters to be learned.

In the next experiment, we used the Principal Component Analysis [46] to reduce the data's dimension and examine the learning ability of the LMNN-based model for various dimensions of the feature vectors. We defined *performance improvement* expressed in percent as

$$100\,(1 - \frac{ERR_{lmnn}}{ERR_{euclid}}), \tag{20}$$

where ERR_{lmnn} denotes the error rate of the KNN classifier ($k = 3$) using the Mahalanobis distance learned by LMNN, and ERR_{euclid} denotes the error rate for the (non-learned) Euclidean distance.

Our fixed-size dataset consisted of 50,000 training samples and 21,430 testing samples. Figure 7 illustrates *performance improvement* for the LMNN-based model. The result of linear regression represented by the red dashed line shows that the improvement of error rate declines with increasing dimension. This result may indicate that for higher dimensions, the size of our dataset may be a limiting factor.



(a) Dataset A (error rate = 3.29%)

(b) Transformed dataset A by LMNN that was trained on dataset A (error rate = 1.43%)

(c) Dataset B (error rate = 3.14%)

(d) Transformed dataset B by LMNN that was trained on dataset A (error rate = 2.29%)

FIGURE 6. Visualization of the impact on similarities of samples achieved by LMNN using the t-SNE algorithm. Red crosses represent malicious files, while blue dots represent benign files. There are two different datasets, both consisted of 700 samples. For dataset A transformed by LMNN that was trained on the same dataset A we achieved the error rate of 1.43 %, while for dataset B transformed by LMNN that was trained on the different dataset we achieved an error rate of 2.29 %. All results were achieved using the KNN classifier (k = 3).

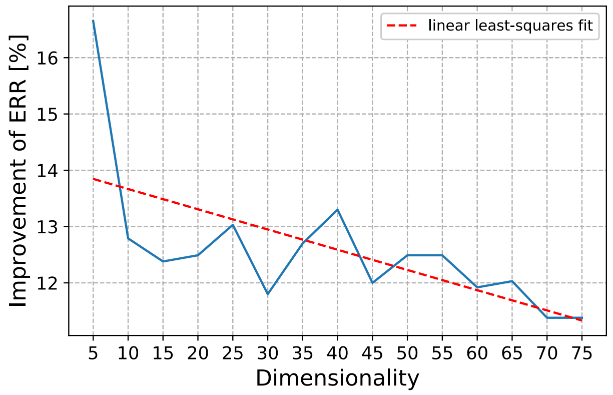


FIGURE 7. The relation between the dimensionality of the feature vector and *performance improvement* achieved using the KNN classifier with Mahalanobis distance metric learned by LMNN.

C. MINIMIZING OF THE FALSE POSITIVE RATE

This section concerns the problem of detecting as much malware as possible while maintaining a low false positive rate. We first focus on minimizing the false positive rate using the PSO algorithm. We analyzed how the coefficient c in the WERR criterion defined in Eq. (15) influences the false positive rate and the error rate.

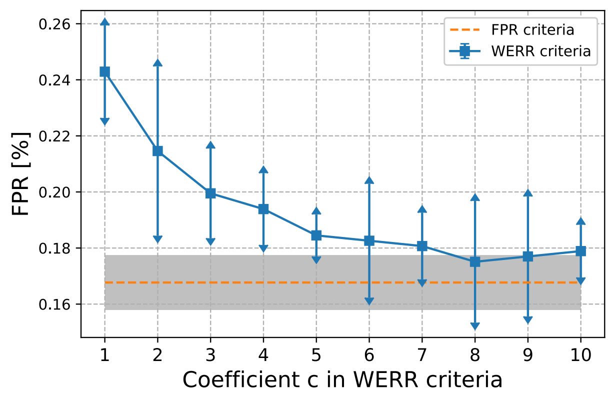
TABLE 4. Results of the experiment with the modified LMNN and the PSO-based method with emphasis on low false positive rate.

Method	TPR[%]	FPR[%]	ERR[%]
PSO with WERR	97.87	0.13	1.15
modified LMNN	97.50	0.19	1.42

In this experiment, we performed PSO with WERR optimization criterion for $c \in \{1, \dots, 10\}$. The relation between the coefficient c and the false positive rate and the error rate achieved by the KNN classifier ($k = 3$) is presented in Figure 8. The PSO was performed ten times for randomly chosen 50,000 training samples and 21,430 testing samples. The figure shows the mean values of FPR and ERR with the standard deviation.

As expected, with increasing coefficient c the corresponding FPR decreases. However, for $c > 8$, FPR does not decrease anymore since KNN using the Mahanalobis distance produces only 20 to 30 false positives. Given the size of our dataset, the lowest FPR, 0.13 %, was achieved for $c = 8$. Note that with increasing coefficient c the corresponding ERR increases as well while $c \leq 8$.

While 0.13% FPR with 1.15 % error rate achieved in our experiment seems reasonable, it can still be impractical in real-world applications. It is undesirable that antivirus programs would delete a benign sample once in every



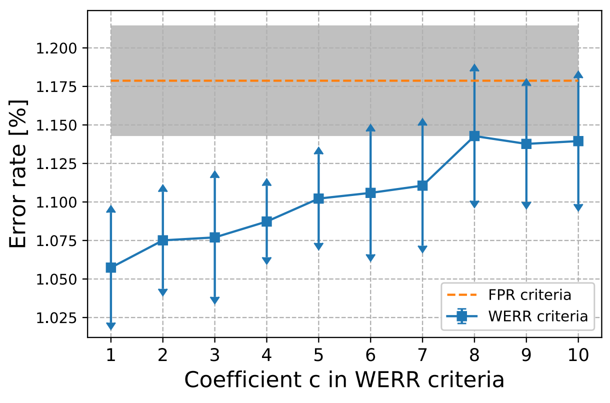


FIGURE 8. The relation between the coefficient c defined in the WERR criterion and false positive rate and error rate.

TABLE 5. List of machine learning classifiers with the corresponding names of algorithms from the Scikit-learn library and the corresponding parameters. For some classifiers, we used parameters with default values as stated in the Scikit-learn library.

Name of classifier	Name of algorithm in Scikit-learn	Parameters
k -Nearest Neighbor	`neighbors.KNeighborsClassifier`	n_neighbors = 3
Support Vector Machine	`svm.SVC`	default parameters
Logistic Regression	`linear_model.LogisticRegression`	default parameters
Naïve Bayes	`naive_bayes.GaussianNB`	default parameters
Decision Tree	`tree.DecisionTreeClassifier`	default parameters
Deep Neural Network	`neural_network.MLPClassifier`	hidden_layer_sizes=(200,100), max_iter=300, activation = relu, solver=adam
Ada Boost	`ensemble.AdaBoostClassifier`	n_estimators=100
Random Forest	`ensemble.RandomForestClassifier`	n_estimators=100

769 scanned samples on average. However, our proposed malware detection model can be used as one component of a more complex system relying on more data types from both the static and the dynamic analysis.

As for minimizing FPR using LMNN, we modified Eq. (6) by adding the parameter η_k which corresponds to the cost of false positive. This modification aims to minimize the number of impostors belonging to the class of benign files. Let T be a training set and let N_i denote the set of k target neighbors of x_i. Then the modification of LMNN focusing on minimizing the false positives is:

$$\min_{\mathbf{L}} \sum_{x_i \in T} \sum_{x_j \in N_i} \Big(d_L(x_i, x_j)^2 + \mu \sum_{x_k \in T: y_i \neq y_k} \eta_k [1 + d_L(x_i, x_j)^2 - d_L(x_i, x_k)^2]_+ \Big) \tag{21}$$

where η_k denotes the cost of false positive and it is defined as:

$$\eta_k = \begin{cases} 1 & \text{if } y_k \text{ is class of benign files,} \\ c \geq 1 & \text{if } y_k \text{ is class of malware.} \end{cases}$$

Similarly to the WERR optimization criterion, the purpose of the parameter c in the definition of η_k is to set the amount of penalization for one false positive. The difference between the modification of LMNN and WERR criterion is that the modification of LMNN takes into account the distance between a sample and its impostor.

To summarize the result, Table 4 shows the performance of the modified LMNN according to Eq. (21) and the PSO methods with the WERR criterion.

TABLE 6. Averaged classification results.

ML Method	TPR[%]	FPR[%]	ERR[%]
our proposed method	98.64	**0.74**	**1.09**
k-Nearest Neighbor	98.21	1.11	1.45
Support Vector Machine	98.20	1.76	1.78
Logistic Regression	98.22	1.82	1.80
Naïve Bayes	87.59	1.51	7.63
Decision Tree	98.26	1.70	1.72
Deep Neural Network	**98.87**	1.45	1.30
Ada Boost	98.66	2.07	1.71
Random Forest	98.49	1.00	1.26

Note that the PSO-based method resulted in a lower false positive rate when compared to the LMNN-based method.

D. COMPARISON TO THE STATE-OF-THE-ART MACHINE LEARNING ALGORITHMS

In the last experiment, we compared several state-of-the-art machine learning algorithms with *our proposed method* which refers to the KNN classifier with weighted Euclidean distance where the weights were learned by the PSO algorithm with WERR criterion.

A list of machine learning classifiers considered, together with implementation details, is presented in Table 5. We briefly describe the machine learning techniques applied in the experiment.

The k-Nearest Neighbors classifier [47] is one of the most popular supervised learning methods. It is a non-parametric method that assigns a class label to each tested sample by a majority vote of its k nearest neighbors.

Support Vector Machine method (SVM) [48] is mainly defined for two-class classification problems. The core idea

TABLE 7. List of 75 features selected by RFE Logistic Regression algorithm. Some numeric fields were considered as nominal since they have a low number of different values.

Field	Structure	# features	type
NumberOfSymbols	COFF File Header	1	numeric
NumberOfSections	COFF File Header	1	numeric
AddressOfEntryPoint	Optional Header Standard Fields	1	numeric
MajorLinkerVersion	Optional Header Standard Fields	1	nominal
MinorLinkerVersion	Optional Header Standard Fields	1	nominal
SizeOfHeapCommit	Optional Header Windows-Specific Fields	1	numeric
ImageBase	Optional Header Windows-Specific Fields	1	nominal
SectionAlignment	Optional Header Windows-Specific Fields	1	nominal
FileAlignment	Optional Header Windows-Specific Fields	1	nominal
MajorOperatingSystemVersion	Optional Header Windows-Specific Fields	1	nominal
MajorImageVersion	Optional Header Windows-Specific Fields	1	nominal
MajorSubsystemVersion	Optional Header Windows-Specific Fields	1	nominal
MinorSubsystemVersion	Optional Header Windows-Specific Fields	1	nominal
SizeOfImage	Optional Header Windows-Specific Fields	1	numeric
CheckSum	Optional Header Windows-Specific Fields	1	nominal
Subsystem	Optional Header Windows-Specific Fields	1	nominal
NumberOfRvaAndSizes	Optional Header Windows-Specific Fields	1	nominal
RVA all 16 fields	Optional Header Data Directories	16	numeric
Size all 16 fields	Optional Header Data Directories	16	numeric
PointerToRawData	Section Header - all 5 sections	5	numeric
SizeOfRawData	Section Header - all 5 sections	5	numeric
VirtualAddress	Section Header - all 5 sections	5	numeric
VirtualSize	Section Header - all 5 sections	5	numeric
Characteristics - IMAGE_SCN_TYPE_DSECT	Section Flags for the first section	1	boolean
Characteristics - IMAGE_SCN_TYPE_COPY	Section Flags for the first section	1	boolean
Characteristics - IMAGE_SCN_TYPE_NOLOAD	Section Flags for the first section	1	boolean
Characteristics - IMAGE_SCN_TYPE_GROUP	Section Flags for the first section	1	boolean
Characteristics - IMAGE_SCN_TYPE_REG	Section Flags for the first section	1	boolean
file_size	not part of PE format	1	numeric

is to maximize the margin, which is the smallest distance between the training data and the decision boundary. The SVM method can also be applied in multiclass classification problems using a binary classifier in a one-against-all situation.

Logistic Regression [49] is a parametric binary classifier that estimates the coefficients from the training data using the maximum-likelihood estimation. Similar to SVM, the One-against-all strategy can also be applied to multiclass classification problems.

The Naïve Bayes classifier [50] is a probabilistic algorithm based on Bayes' theorem that predicts the class with the highest a posteriori probability. The Naïve Bayes classifier is based on the assumption that the features are conditionally independent of one another, which is often not valid in practice.

The Decision Tree classifier [51] is represented as a tree where the internal nodes correspond to features and the leaf nodes correspond to class labels. Edges leading to children node correspond to the feature values. The feature vector determines the path from the root node to the leaf node.

Deep Neural network [52] is a feed-forward artificial neural network that consists of three types of interconnected layers of perceptrons. The input layer takes a feature vector, which is then processed in hidden layers, and finally, perceptrons in the output layer output a result.

Adaboost [53] is one of the most popular boosting algorithms. It runs several weak classifiers and assigns them weights that are based on the corresponding error rates. These weights are then used to predict the output class.

Random forest [54] is an ensemble learning method that combines the results made by several decision trees using a voting mechanism.

Table 6 provides average classification results of the selected supervised machine learning algorithms compared with the results of *our proposed method* defined as the KNN classifier using the weighted Euclidean distance learned by the PSO algorithm as described in Section IV.

All machine learning algorithms were run 20 times on a randomly chosen training and testing set with 50,000 samples and 21,430 samples, respectively. *Our proposed method* outperformed all the machine learning classifiers achieving the lowest FPR and the lowest error rate. Deep Neural Network and Ada Boost were the only ML algorithms having a higher TPR than the PSO-based model; however, they both achieved a significantly higher FPR.

VII. CONCLUSION

This paper proposed a malware detection system based on the k-Nearest Neighbor classifier using the weighted Euclidean distance learned by the Particle Swarm Optimization algorithm. We empirically demonstrated that our approach achieved the lowest error rate and the lowest false positive rate among all state-of-the-art machine learning algorithms considered in our experiment. We described the architecture of the detection system based on structural information from the static analysis of Windows PE files. This approach can also be applied to executable formats of other operating systems, such as macOS or Linux.

In addition, we focused on the problem of detecting as much malware as possible while keeping a low false positive rate because a high false positive error is considered seriously in the antivirus industry. We proposed an optimization criterion based on a weighted error rate to penalize false positives. Using this criterion as a fitness function in the Particle Swarm Optimization algorithm, which was used to learn the feature weights of the weighted Euclidean distance, we achieved 0.13 % false positive rate with an error rate of 1.15 %.

Ongoing work is focused in two directions. First, we are working on learning multiple local distance metrics for different malware families. We plan to investigate both unsupervised and supervised methods. Secondly, it would be interesting to experiment with other distance metric learning algorithms with various optimization criteria to achieve an even lower FPR with an acceptable error rate.

APPENDIX A
FEATURES USED IN EXPERIMENTS

Table 7 summarizes the list of 75 features all extracted from the PE file format. For each feature from a section header, we considered the order of the section rather than the name of the section (such as.text,.data,.rsrc). While the sections' order turns out to be important for malware detection, this kind of information is often not mentioned in research papers. We keep the name of the fields in the same form as in the documentation [43] so that the reader can easily find a detailed description in the documentation [6].

REFERENCES

[1] D. Salomon, *Foundations of Computer Security*. London, U.K.: Springer, 2006.

[2] K. Monnappa, *Learning Malware Analysis: Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware*. Birmingham, U.K.: Packt Publishing Ltd, 2018.

[3] J. Singh and J. Singh, "A survey on machine learning-based malware detection in executable files," *J. Syst. Archit.*, vol. 112, Jan. 2021, Art. no. 101861.

[4] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Abu Dhabi, UAE: Lulu, 2013.

[5] M. Jureček, O. Jurečková, and R. Lórencz, "Improving classification of malware families using learning a distance metric," in *Proc. 7th Int. Conf. Inf. Syst. Secur. Privacy*, 2021, pp. 643–652.

[6] Microsoft. *PE Format—Win32 Apps*. Accessed: May 29, 2021. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format

[7] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is Packin' heat; limits of machine learning classifiers based on static analysis features," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–20.

[8] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526.

[9] O. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[10] M. Wadkar, F. Di Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 113022.

[11] IMDEA Software Institute. *MALICIA PROJECT Malware Cybercrime*. Accessed: May 29, 2021. [Online]. Available: https://malicia-project.com

[12] *VirusShare*. Accessed: May 29, 2021. [Online]. Available: https://virusshare.com

[13] L. Yang and J. Liu, "TuningMalconv: Malware detection with not just raw bytes," *IEEE Access*, vol. 8, pp. 140915–140922, 2020.

[14] *MalShare*. Accessed: May 29, 2021. [Online]. Available: https://malshare.com

[15] X. Gao, C. Hu, C. Shan, B. Liu, Z. Niu, and H. Xie, "Malware classification for the cloud via semi-supervised transfer learning," *J. Inf. Secur. Appl.*, vol. 55, Dec. 2020, Art. no. 102661.

[16] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018, *arXiv:1802.10135*. [Online]. Available: https://arxiv.org/abs/1802.10135

[17] D. Xue, J. Li, T. Lv, W. Wu, and J. Wang, "Malware classification using probability scoring and machine learning," *IEEE Access*, vol. 7, pp. 91641–91656, 2019.

[18] *VX Heavens*. Accessed: May 29, 2021. [Online]. Available: https://83.133.184.251/virensimulation.org/index.html

[19] W. Zhong and F. Gu, "A multi-level deep learning system for malware detection," *Expert Syst. Appl.*, vol. 133, pp. 151–162, Nov. 2019.

[20] *Maltrieve*. Accessed: May 29, 2021. [Online]. Available: https://github.com/krmaxwell/maltrieve

[21] *Offensive Computing*. [Online]. Available: https://www.offensivecomputing.net

[22] *VirusSign*. Accessed: May 29, 2021. [Online]. Available: https://www.virussign.com

[23] E. Raff, J. Sylvester, and C. Nicholas, "Learning the pe header, malware detection with minimal domain knowledge," in *Proc. 10th ACM Workshop Artif. Intell. Secur.*, 2017, pp. 121–132.

[24] *Open Malware*. Accessed: May 29, 2021. [Online]. Available: https://openmalware.org

[25] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3838–3845.

[26] A. Kumar, K. S. Kuppusamy, and G. Aghila, "A learning model to detect maliciousness of portable executable using integrated feature set," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 31, no. 2, pp. 252–265, Apr. 2019.

[27] M. Jureček and R. Lórencz, "Distance metric learning using particle swarm optimization to improve static malware detection," in *Proc. 6th Int. Conf. Inf. Syst. Secur. Privacy*, 2020, pp. 725–732.

[28] M. Jureček and R. Lórencz, "Malware detection using a heterogeneous distance function," *Comput. Informat.*, vol. 37, no. 3, pp. 759–780, 2018.

[29] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *J. Artif. Intell. Res.*, vol. 6, pp. 1–34, Jan. 1997.

[30] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artif. Intell. Rev.*, vol. 11, nos. 1–5, pp. 273–314, 1997.

[31] J. L. Suárez, S. García, and F. Herrera, "A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges," *Neurocomputing*, vol. 425, pp. 300–322, Feb. 2021.

[32] L. Yang and R. Jin, "Distance metric learning: A comprehensive survey," *Michigan State Univ.*, vol. 2, no. 2, p. 4, 2006.

[33] B. Kulis, "Metric learning: A survey," *Found. Trends Mach. Learn.*, vol. 5, no. 4, pp. 287–364, 2013.

[34] A. Bellet, A. Habrard, and M. Sebban, "A survey on metric learning for feature vectors and structured data," 2013, *arXiv:1306.6709*. [Online]. Available: https://arxiv.org/abs/1306.6709

[35] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 1473–1480.

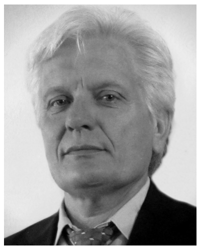
[36] K. Q. Weinberger and L. K. Saul, "Fast solvers and efficient implementations for distance metric learning," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 1160–1167.

[37] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," in *Proc. Adv. Neural Inf. Process. Syst.*, 2005, pp. 513–520.

[38] K. Q. Weinberger and G. Tesauro, "Metric learning for kernel regression," in *Artificial Intelligence and Statistics*. San Juan, Puerto Rico: PMLR, 2007, pp. 612–619.

[39] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, Nov. 1995, pp. 1942–1948.
[40] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, vol. 26. New York, NY, USA: Springer, 2013.
[41] E. Carrera. *Pefile*. Accessed: May 29, 2021. [Online]. Available: https://github.com/erocarrera/pefile
[42] Scikit-Learn. *Scikit-Learn: Machine Learning in Python*. Accessed: May 29, 2021. [Online]. Available: https://scikit-learn.org
[43] *Microsoft Portable Executable and Common Object File Format Specification*, Microsoft, Washington, DC, USA, 1999.
[44] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: An overview," *Soft Comput.*, vol. 22, no. 2, pp. 387–408, Jan. 2018.
[45] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.
[46] L. I. Smith, "A tutorial on principal components analysis," Univ. Otago, Dunedin, New Zealand, Tech. Rep. OUCS-2002-12, 2002.
[47] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.
[48] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Comput. Learn. Theory (COLT)*, Jul. 1992, pp. 144–152.
[49] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, vol. 1, no. 10. New York, NY, USA: Springer, 2001.
[50] M. E. Maron and J. L. Kuhns, "On relevance, probabilistic indexing and information retrieval," *J. ACM*, vol. 7, no. 3, pp. 216–244, Jul. 1960.
[51] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.
[52] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
[53] R. E. Schapire and Y. Freund, "Boosting: Foundations and algorithms," *Kybernetes*, vol. 42, no. 1, pp. 1–35, Jan. 2013.
[54] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

MARTIN JUREČEK received the degree from the Faculty of Mathematics and Physics, Charles University in Prague, with a specialization in mathematical methods of information security. He is currently pursuing the Ph.D. degree with the Faculty of Information Technology, Czech Technical University in Prague. His research interests include the application of machine learning, artificial intelligence approaches to malware detection, algebraic cryptanalysis, and the security of cryptocurrencies.



RÓBERT LÓRENCZ received the degree from the Faculty of Electrical Engineering, Czech Technical University in Prague, in 1981, and the Ph.D. degree from the Institute of Measurement and Measuring Methods, Slovak Academy of Sciences, Bratislava. He is currently a Full Professor with the Faculty of Information Technology, Czech Technical University in Prague. His research interests include cryptography and arithmetic units for cryptography primitives, various cryptanalysis methods of block, stream ciphers, and alternative arithmetic for numerical computation.