FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Automatic Malware Detection

by

Martin Jureček

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Department of Information Security

Prague, March 2021

Supervisor:
prof. Ing. Róbert Lórencz, CSc.
Department of Information Security
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

The problem of automatic malware detection presents challenges for antivirus vendors. Since the manual investigation is not possible due to the massive number of samples being submitted every day, automatic malware classification is necessary.

Our work is focused on an automatic malware detection framework based on machine learning algorithms. We proposed several static malware detection systems for the Windows operating system to achieve the primary goal of distinguishing between malware and benign software. We also considered the more practical goal of detecting as much malware as possible while maintaining a sufficiently low false positive rate.

We proposed several malware detection systems using various machine learning techniques, such as ensemble classifier, recurrent neural network, and distance metric learning. We designed architectures of the proposed detection systems, which are automatic in the sense that extraction of features, preprocessing, training, and evaluating the detection model can be automated. However, antivirus program relies on more complex system that consists of many components where several of them depends on malware analysts and researchers.

Malware authors adapt their malicious programs frequently in order to bypass antivirus programs that are regularly updated. Our proposed detection systems are not automatic in the sense that they are not able to automatically adapt to detect the newest malware. However, we can partly solve this problem by running our proposed systems again if the training set contains the newest malware.

Our work relied on static analysis only. In this thesis, we discuss advantages and drawbacks in comparison to dynamic analysis. Static analysis still plays an important role, and it is used as one component of a complex detection system.

Keywords:
Malware, PE File Format, Static Analysis, Machine Learning, Detection System.

Acknowledgements

I would like to express my gratitude to my dissertation thesis supervisor, prof. Ing. Róbert Lórencz, CSc., for his guidance and for having always encouraged me and supported my research. I am also very grateful to my colleagues from the Department of Information Security, who shared their research ideas and experiences from the doctoral study.

This research has also been partially supported by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics".

Finally, my greatest thanks go to my wife for her infinite patience and care and to my daughter, who gave me the motivation to finish the doctoral study.

Dedication

. . .

To my wife Olga and my daughter Dáša.

Contents

List of Figures

List of Algorithms

Abbreviations

ACC	Accuracy
API	Application Programming Interface
APK	Android Package
AUC	Area Under Curve
AV	Antivirus Vendor
BLSTM	Bidirectional Long Short-Term Memory
COFF	Common Object File Format
DLL	Dynamic-Link Library
DML	Distance Metric Learning
DDoS	Distributed Denial-of-Service
DT	Decision Tree
ERR	Error rate
FPR	False Positive Rate
GR	Gain Ratio
KNN	k-Nearest Neighbor
LMNN	Large Margin Nearest Neighbor
LSTM	Long Short-Term Memory
ML	Machine Learning
NB	Naive Bayes
NCA	Neighborhood Components Analysis
MLKR	Metric Learning for Kernel Regression
PAM	Partitioning around medoids
PE	Portable Executable
PUP	Potentially Unwanted Program
ROC	Receiver Operating Characteristic
SC	Silhouette Coefficient
SVM	Support Vector Machine
TFIDF	Term Frequency-Inverse Document Frequency
TPR	True Positive Rate
VDM	Value Difference Metric
WERR	Weighted Error Rate
WKNN	Weighted k-Nearest Neighbor

Chapter 1

Introduction

In information security, malware attacks are 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, on other hand, malware analysts and researchers continually improve their defenses. This game seems to have an infinite number of rounds.

The attackers purpose is no longer to cause detriment, such as damaging a computer system. 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 [1], [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.

Automated malware detection based on machine learning consists of extraction of useful information from training samples, application of data preprocessing techniques, feature selection, building a detection model, and finally evaluation of the performance. Since malware evolves, this whole process runs repeatedly.

1.1 Motivation

As the number of malware samples increases every day, the need for automatic classification of malware that would be able to learn and adapt is crucial. According to the AV-TEST Security Report 2019/2020 [3], more than 114 million new malware were developed in 2019. To defend against malware, users typically rely on antivirus products to detect a threat before it can damage their systems. Antivirus vendors rely mainly on a database of sequences of bytes (signatures) that uniquely identify the suspect files and are unlikely to be found in benign programs.

The major weakness of signature detection is that malware writers can easily modify their code, thereby changing their program's signature and evading virus scanners. The

signature detection technique is unable to detect obfuscated and zero-day malware.

Encryption, polymorphism, metamorphism, and other code obfuscation techniques are widely used by malware authors to evade signature detection techniques. For this reason, malware researchers are investigating novel detection strategies.

During the last years, the current trend is to used a malware detection framework based on machine learning algorithms [4], [5]. Thanks to cloud-based computing, which makes the cost of big data computing more affordable, the concept of employing machine learning for malware detection has become more realistic to deploy.

1.2 Problem Statement

Malware detection is one of the most important problems since the detection of malware in advance allows us to block it. Malware detection is a binary classification problem of distinguishing between malware and benign files. Malware family classification is a multiclass classification problem where each testing sample is assigned to a known malware family. The practical use of distinguishing between malware families lies in helping malware analysts to deal with a large number of samples.

One of the main problems of malware detection systems is insufficient accuracy while keeping the false positive rate at an acceptable level. There is a need to build a machine learning framework suited for real-life practical use that generically detects as many malware samples as possible, with a very low false positives rate. The significant problem to be solved is how to detect malware that has never been seen before. While signature-based detection systems identify known malicious programs, they can be bypassed by unknown malware. Instead of using static signatures, an adequate alternative solution is to use machine learning methods to detect malware.

1.3 Goals of the Dissertation Thesis

In this thesis, we attempt to reach the following four goals via machine learning techniques.

1. To detect malware with a minimal error rate. More precisely, to achieve the error rate of less than 1%.

2. To detect as much malware as possible while maintaining a low false positive rate. More precisely, to achieve the error rate and false positive rate, both of less than 1%.

3. To experimentally verify the usefulness of distance metric learning for malware detection and classification of malware families.

4. To automate the process of malware detection and malware family classification.

1.4 Structure of the Dissertation Thesis

The dissertation thesis is organized into the following five chapters:

1. *Introduction* describes the motivation behind our efforts, defines problem statement, and lists the goals of this dissertation thesis.
2. *Background and State-of-the-Art* introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.
3. *Overview of Our Approach* summarizes author's work in the field of automatic malware detection.
4. *Author's Relevant Papers* presents a collection of five author's papers relevant to this thesis.
5. *Conclusions* summarizes the results of our research, suggests possible topics for further research, and concludes the thesis.

Chapter 2

Background and State-of-the-Art

2.1 Background

The term malware, or malicious software, is defined as any software that does something that causes detriment to the user. Malware includes viruses, worms, trojan horses, rootkits, spyware, and any other program that exhibits malicious behavior. The attacker's purpose is no longer either to infiltrate or to damage a computer system. Nowadays, malware has become a rather profitable business.

Malware writers usually employ obfuscation techniques such as encryption, binary packers, or self-modifying code to evade malware classifiers.

Malware detection techniques can be typically classified into two categories depending on how code is analyzed: static and dynamic analysis. The static analysis aims at searching for information about the structure and data in the file. The disassembly technique is a static analysis technique, which is used to extract various features from the executables. The dynamic analysis aims to examine a program that is executed in a real or virtual environment. Many malware researchers have focused on machine learning algorithms to defeat obfuscation techniques and detect unknown malware.

The choice of the most suitable machine learning algorithm is affected by the types of features extracted from samples and their representation. This section presents several types of features extracted from static or dynamic analysis.

2.1.1 Types of Malware

The term malware is short for malicious software, and it refers to any software that does something that causes detriment to the user. The problem with each definition of malware is that it is very broad and not rigorous. Malicious activities may include disrupt a computer operation, gather sensitive information, or unauthorized access to a computer system or network.

To the best of our knowledge, there is no definition of what precisely malware is. This lack causes practical consequences in the antivirus industry and poses the following

question. If we do not know the exact definition of malware, then is it possible to create some malware detection models with 100% of accuracy? Malware classification models are evaluated using labeled data from the test set. However, there is an assumption that the testing data are labeled correctly. If we want to achieve perfect accuracy for each user of some antivirus program, there must be a consensus among all users on the definition of malware. Since there is no such exact definition, the evaluation results of each malware detection model are only relative to the labels of samples from the test set. An example of programs where consensus is not achieved is adware which is sometimes referred to as malware and sometimes as benign files.

Malware classification can take into account the purpose of malware or its functionality. However, classification based on malware's functionality may not be possible since malware can have many functionalities. For example, malware can behave like a worm (i.e., scans the network and exploits vulnerabilities) and can download another malware component such as backdoor [1]. As a result, the following types of malware are not mutually exclusive, and they are solely intended to familiarize the reader with the various types of malware.

Backdoor is classified as a Trojan horse, and it enables the attacker to get access to a system and execute commands.

Bot is malware that allows the attacker (called botmaster) to control the compromised system remotely. A group of interconnected bots remotely-controlled by a botmaster using Command and Control (C&C) software [1] is called a botnet. Usual malicious activities of a botnet are Distributed Denial-of-Service (DDoS) attacks, spyware activities, sending spam emails, or distributing other malware.

Downloader also called dropper, is designed to download and install additional malware or malicious components.

Ransomware is type of malware that locks victim's screen and encrypts chosen types of files (such as popular .xsl, .docx, .txt, .sql, .jpg, .cpp, .mp3, and many others). The AES, RSA, and Elliptic curve cryptography are the most common encryption transformations. The attacker then demands a ransom (usually paid in Bitcoin [6]) for providing the decryption key.

Rootkit is used to modify the operating system in order for an attacker can keep administrator privileges. The characteristic of rootkit is to conceal its presence or the presence of other malicious programs.

Spyware retrieves sensitive data from a victim's system and sends them to the attacker. Such sensitive data may include passwords, credit card numbers, history of visited web pages, emails, and various documents. An example of spyware is keylogger [1] that captures keystrokes and transfer them to the attacker. Another example of spyware is a sniffer that monitors internet traffic [8].

[1]Note that keystroke dynamics can be used to identify a user in the system [7], and as a result, increase the security of the system.

Trojan horse disguises itself as benign software; however, it performs malicious activities in the background.

Virus is malware that is attached to a host file. Virus depends on human activity, and when such an infected host is run, the virus is activated and performs some malicious activity, and spread itself to other computers. A comprehensive discussion on computer viruses can be found in [9].

Worm is similar to a virus; however, it does not need a host file nor human activity to activate itself. A worm can self-replicate and spreads itself to other computers.

Potentially Unwanted Programs (PUP) are a specific type of software that is considered as a gray area among malware and benign files. The intent of PUP may be unclear, or for some users, the benefit of PUP may outweigh the potential risk. Antivirus vendors deal with PUPs in with lower-risk category. Some antivirus vendors allow users to decide whether PUPs will be detected or not on their system. An example of PUP is an adware that displays advertisements, often in the form of pop-up messages. Another example of PUP is toolbars, extensions, or plugins installed on the user's browser. Note that adware is often determined as malware, however, this statement is dependent on the definition of malware which is not fixed.

Malware writers use a variety of infection vectors to distribute malicious programs and infect devices. They can exploit vulnerable services over the network, vulnerabilities in the Web browser application, or social engineering. Popular techniques used to spread malware to users are presented in [10].

Malware authors usually use various obfuscation techniques to avoid malware detection. The following section presents the most used techniques to evade detection from anti-virus products.

2.1.2 Malware Obfuscation Techniques

Malware authors often use techniques to modify malware to make it more difficult to detect it. These techniques are called obfuscation techniques, and they are used to protect the malware from malware analysts and reverse engineers. These techniques successfully avoid signature-based detection. Obfuscation can be applied on several layers, such as code, a sequence of instructions, or binary. In this section, we will outline some techniques used to make it more difficult to detect malware. Among the most popular techniques belong packing, encryption, polymorphism, and metamorphism.

Packing is a process that uses compression (one or more layers) to obfuscate an executable file. As a result, a new executable file with obfuscated content is created. The unpacking process consists of a decompression routine that retrieves the original file (or code).

Encryption is similar to packing; however, encryption is used instead of compression. Encrypted and packed malware must contain a decryption module, which can be used for signature-based detection.

Polymorphism uses encryption, and in addition, the decryptor module is morphed, and as a result, exhibits no signature. However, polymorphic malware still needs to be decrypted, and original (non-obfuscated) code can be used for signature-based detection.

Metamorphism is the most advanced obfuscation technique and the most challenging for malware authors. Metamorphic malware changes internal structure while maintaining its original functionality.

Packing and encryption are two popular techniques to avoid signature-based detection and static analysis. Polymorphic and metamorphic malware have the ability to change their code with each new generation. When such kind of malware is executed, it can be obfuscated again to avoid signature-based detection.

2.1.3 Malware Analysis

The purpose of malware analysis is to provide the information used in classification or clustering problems.

2.1.3.1 Static vs. Dynamic Analysis

Malware detection techniques can be classified into two main categories depending on how code is analyzed: static and dynamic analysis. Static analysis [11], [12] aims at searching information about structure of a file. The disassembly technique is one technique of static analysis, which is used to extract various features from the executables. Dynamic analysis [13], [10] aims to examine a program which is executed in a real or virtual environment.

Since dynamic analysis involves running the program, information from this kind of analysis is more relevant than information from static analysis.

Several works [14], [15], [16], [17] have described various limitations of static analysis. The most important drawback is that data captured from the 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, the static analysis still has its place in malware detection systems.

A combination of static analysis and dynamic analysis is called hybrid analysis. Several works [18], [19] demonstrated that hybrid analysis has the potential to leverage advantages from both static and dynamic analysis.

2.1.3.2 Signatures vs. Machine Learning Approaches

Signatures
Most AV vendors rely primarily on the signature detection technique, a relatively simple and efficient rule-based method for detecting known malware [20]. Signature (unique hex code strings) of the malware is extracted and added to the database. The antivirus engine compares the file contents with all malware signatures in the database, and if a match is found, the file is reported as malware. A good signature must capture malware with minimal false positive probability. The major weakness of signature detection is its inability to detect obfuscated and zero-day malware. Another drawback is that human analysts help create signatures that result in prone to errors.

Non-Signature Techniques based on Machine Learning Methods
Machine learning methods can partly solve the problem of an inability to detect unknown malware. They can detect previously unknown or obfuscated malware, and they do not need humans to create any kind of signature. However, ML methods are prone to have a high false positive rate and have large processing overheads.

Machine learning algorithms can be divided into three main categories:

Supervised learning - labeled data are utilized to gain knowledge that is used to predict labels of new samples.

Unsupervised learning - unlabeled input data are utilized to acquire knowledge.

Semi-supervised learning - labeled and unlabeled data are inputs to a statistical model. Unlabeled data with a small amount of labeled data can improve the accuracy of the model.

The workflow of ML-based malware detection is an iteration process and consists of the following five steps:

1. **Feature extraction (static and/or dynamic)**. The process of feature extraction is performed through either static and dynamic analysis, and it depends on the types of features.

2. **Feature preprocessing**. Feature extracted from the previous step can have various representations. Depending on the data representation, features can be normalized, encoded to a more appropriate representation, missing values can be filled by average values, and noisy values can be removed.

3. **Feature selection**. After preprocessing step, a number of features can be reduced to improve performance. A subset of the most relevant features can be selected, or original data can be transformed into a reduced number of features.

4. **Machine learning algorithms**. Malware detection problems are usually defined as classification or clustering problems. The choice of ML algorithms depends on whether training data is labeled or not, as we mentioned above, and this choice depends on data representation as well.

5. **Evaluation of performance**. Performance metrics are used to evaluate ML algorithms. Evaluation results indicate how good results were achieved and which ML algorithm performed better with respect to a given dataset.

More information on feature selection and preprocessing as well as evaluation metrics can be found in [21]. The works [22], [23], [24] survey the most popular machine learning techniques for malware classification in the Windows operating system. Several types of features are described in the following section.

2.1.4 Features for Malware Detection

A program can be represented on various levels of abstraction. Static and dynamic analysis techniques can be applied to different representations of a program. Some types of features can be extracted from both static and dynamic analysis. For example list of Application Programming Interface (API) functions is included in the Portable Executable (PE) file without running the program. On the other hand, API calls can be extracted dynamically during the execution of the program, which could be significantly more time-consuming.

Byte sequences - static analysis - byte sequences are extracted from a program that is treated as a sequence of bytes. Examples of bytes sequence can be found in [25]. The most popular method of using byte sequences or opcode sequences is based on frequency distribution, such as the n-gram method [25].

API & system calls - static or dynamic analysis - API functions and system calls provide information on the program's behavior related to file systems, networking, security, and other resources provided by the operating system.

opcodes - static analysis - opcode (operation code) sequences are extracted from the assembly language source code. Common techniques are based on the frequency of appearance of opcode-sequences [26], examination of opcode frequency distribution difference between malicious and benign code [27], or identification of critical instruction sequences [28].

PE file characteristics - static analysis - the features used in our experiments are extracted from the portable executable (PE) file format [29], which is the file format for executables, Dynamis-Link Libraries (DLL), object code, and others used in 32-bit and 64-bit versions of the Windows operating system. 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 Common Object File Format (COFF) file header, the optional header, and the section headers. The optional file header is followed immediately by the section headers, which provide information about sections, including locations, sizes, and characteristics.

Strings - static analysis - printable string extracted from a program can reveal valuable information, such as URLs where the program connects, file names, and file paths.

Entropy - static analysis - compressed or encrypted programs have a higher statistical variation of bytes sequence, and as a result, higher entropy than native code.

Image - static analysis - malwares binary content can be represented as a grayscale image where every byte of a program represents one pixel. The array of pixels is then reorganized to a 2D image.

Instruction Traces - dynamic analysis - a program can be represented as a sequence of processor instructions. While packing and encryption can avoid static analysis of instruction traces, dynamic analysis based on instruction traces bypasses such anti-malware analysis techniques.

Features from the previous list belong among the most used features for malware detection. However, there are several other types of features, such as features extracted from filesystem, memory, network activity [24]. Some features can be represented as a directed graph. For example, function calls are represented as a function call graph, and control flow paths are represented as a control flow graph. Both types of features are described in [30].

2.2 Previous Results and Related Work

In this section, we survey some relevant previous work in the area of malware detection. The application of machine learning techniques to malware detection has been an active research area for about the last twenty years. Researchers have tried to apply various well-known techniques such as Neural Networks, Decision Trees, Support Vector Machines, ensemble methods, and many other popular machine learning algorithms. Recent survey papers [23], [24], [30], [31] provide comprehensive information on malware detection techniques using machine learning algorithms.

We mainly discuss the papers on static malware detection based on machine learning techniques. Then, we briefly discuss various existing works related to malware family classification.

2.2.1 Pioneer Works

Schultz et al. [32] introduced the concept of data mining for detecting previously unknown malware. Their research presented three different static feature sources for malware classi-

fication: information from the portable executable (PE) header, strings, and byte sequences extracted from binaries. These features were used in three different kinds of algorithms: an inductive rule-based learner, a probabilistic method, and a multi-classifier system. A rule induction algorithm called Ripper [33] was applied to find patterns in the dynamic-link library (DLL) data (such as the list of DLLs used by the binary, the list of DLL function calls, and the number of different system calls used within each DLL). The well-known probabilistic method, learning algorithm Naive Bayes, was used to find patterns in the string data and n-grams of byte sequences. Multinomial Naive Bayes algorithm that combined the output of several classifiers reached the highest detection rate of 97.76%. The authors tested data mining methods against standard signatures. Their results indicate that the data mining detection rate of a previously unknown malware was twice as high compared to the signature-based methods.

Kolter and Maloof [34] improved Schulz's third technique by using overlapping byte sequences instead of non-overlapping sequences. They used different kinds of classifiers: naive Bayes, instance-based learner, similarity-based classifier called Term Frequency-Inverse Document Frequency (TFIDF), Support Vector Machine (SVM), Decision Trees (DT), and boosted variants of SVM, DT, and TFIDF. Authors evaluated their classifiers' performance by computing the area under a receiver operating characteristic curve. Boosted Decision tree model (J48) achieved the best accuracy, an area under the ROC curve of 0.996, and outperformed the rest of the classifiers.

2.2.2 Recent Works

Next, we review some most recent works related to malware detection based on machine learning techniques. We mainly focus on works using static analysis of Windows PE files.

Wadkar *et al.* [35] proposed the system based on static features extracted from PE files for detecting evolutionary modifications within malware families. Support Vector Machines (SVM) models were trained over a sliding time window, and the differences in SVM weights were quantified using χ^2 statistic. For most of all 13 malware families considered in the experiments, the system detected significant changes.

Yang and Liu [36] proposed a detection model called TuningMalconv with two layers: raw bytes model in the first layer and 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 in [37]. The detection model is based on semi-supervised transfer learning and was deployed on the cloud as a SaaS (Software as a Service). The detection model was evaluated on Kaggle malware datasets, and it improved classification accuracy from 94.72% to 96.90%.

In [38], the authors 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 k-Nearest Neighbors. The authors performed experiments on more than 170,000 malware samples from 63 malware families and achieved 98.82% of malware classification accuracy.

Zhong and Gu [39] improved the 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 a specific malware family. As a result, the MLDLS 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 work proposed in [40] was from the PE header, more specifically, from MS-DOS, COFF, and Optional Header. Neural networks were learned from raw bytes, which were not parsed to explicit features, and as a result, no preprocessing nor feature engineering was required. More than 400,000 samples were used for training, and the Fully Connected Neural Network model achieved the highest accuracy.

The neural network architecture combining convolutional and feedforward neural layers was proposed in [41]. The authors used only static malware analysis where inputs to feedforward layers were fields of the PE header, while inputs to convolutional layers were assembly opcode sequences. The proposed hybrid neural network achieved 93% on precision and recall.

The work [42] contains three statistical-based scoring techniques: Hidden Markov models, Simple substitution distance, and Opcode graph-based detection. Authors showed that a combination of these scoring techniques with Support Vector Machine yields significantly more robust results than those obtained using any of the individual scores.

In [43], the authors proposed a data structure called aggregation overlay graph that is suitable to compress metadata without any loss of information. A side effect of this structure is that similar files are stored in the same cluster. Besides analyzing PE metadata from the Windows operating system, the work also focuses on Android APK files. Huge dataset consists of 82 million PE files was reduced by over 93%. Clarification is based on the fact that for many malware families, their variants are highly similar to each other. In other words, a significantly high number of features' values remain the same for several malware variants, and the aggregation overlay graph allows to decrease redundancy when storing metadata.

A malware detection system based on a deep neural network was proposed in [44]. The work is based on static features consisted of bin values of a two-dimensional byte entropy histogram, binary's import address table, features extracted from printable sequences of characters, numerical features extracted from binary's portable executable packaging. The authors achieved a 95% detection rate at 0.1% false positive rate based on more than 400,000 samples.

A static malware detection system based on data mining techniques was proposed in [45]. The feature set consists of information in PE header, DLLs, and API functions. Information Gain was used to selecting relevant features, and Principal Component Analysis was used to reduce the dimension of the feature vector. The dataset consists of almost

250,000 Windows programs in PE file format. However, only less than 11,000 samples were benign. For the imbalanced dataset, the detection rate of 99.6% was achieved by the J48 decision tree classifier implemented in the WEKA [46].

A framework, called PE-Miner, for automatic extraction of features from PE file format, was proposed in [47]. The features are then preprocessed, and finally, machine learning techniques are applied to distinguish between malware and benign files. Using the J48 decision tree classifier, the authors achieved more than 99% detection rate with less than 0.5% false alarm rate.

A statistical-based feature extractor applied on structural features of binary PE files was proposed in [48]. After identifying relevant features, a hypothesis-based classification model and machine learning techniques were applied to a collection of more than 22,000 samples. The J84 decision tree classifier achieved the highest accuracy.

In [49], the authors proposed the Intelligent Malware Detection System (IMDS) that is based on window APIsequence. The system consists of a PE parser, an Objective-oriented association rule generator, and a rule-based classifier. The experimental results were evaluated on almost 30,000 executables. The IMDS achieved 93.07% of accuracy and outperformed the J48 decision tree classifier.

The rest of this section briefly reviews the previous research papers on malware family classification related to our work.

In [50], the authors conducted experiments based on byte n-gram features, and they considered 20 malware families. A binary classification was performed on different levels. In the first level, for each of the 20 families, they performed binary classification for 1,000 malware samples from one family and 1,000 benign samples. In the second level, the malware class consists of two malware families; in the third level, the malware class consists of three malware families, and so on up to level 20, where the malware class contains all of the 20 malware families. The authors applied four state-of-the-art machine learning algorithms: KNN, Support Vector Machines, Random Forest, and Multilayer Perceptron. The best classification results (balanced accuracy) were achieved using KNN and Random Forest, over 90% (at level 20), while KNN achieves the most consistent results.

A fully automated system for analysis, classification, and clustering of malware samples was introduced in [51]. This system is called AMAL, and it collects behavior-based artifacts describing files, registry, and network communication, to create features that are then used for classification and clustering of malware samples into families. The authors achieved more than 99% of precision and recall in classification and more than 98% of precision and recall for unsupervised clustering.

In [52], the authors proposed a malware classification system using different malware characteristics to assign malware samples to the most appropriate malware family. The system allows the classification of obfuscated and packed malware without doing any deobfuscation and unpacking processes. High classification accuracy of 99.77% was achieved on the publicly accessible Microsoft Malware Challenge dataset.

The work in [53] presents a classification method based on static (function length frequency and printable sting) and dynamic (API function names with API parameters) features that were integrated into one feature vector. The obtained results showed that in-

tegrating features improved classification accuracy significantly. The meta-Random Forest classifier achieved the highest weighted average accuracy.

Another malware family classification system referred to as VILO is presented in [54]. They used TFIDF-weighted opcode mnemonic permutation features and achieved between 0.14% and 5.42% fewer misclassifications using KNN classifier than does the usage of n-gram features.

Chapter 3

Overview of Our Approach

This chapter describes our contributions that are divided into the following sections. Heterogeneous distance function specially designed for PE the file format and combination of weighted k-Nearest Neighbors classifier and statistical-based classifier are proposed in Section 3.1. In Section 3.2, we modified the Particle Swarm Optimization algorithm and applied it to the problem of finding the most appropriate feature weights used in the heterogeneous distance function. Section 3.3 describes transformation of PE features using various LSTM network architectures. The detection system is based on supervised machine learning algorithms that are applied to the transformed feature space. Section 3.4 focuses on the multiclass classification of six prevalent malware families and benign files. Three state-of-the-art distance metric learning algorithms were employed to learn the Mahalanobis distance metric to improve the performance of k-Nearest Neighbors classifier. Finally, Section 3.5 describes the architecture of our proposed malware detection model using distance metric learning. We focused 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. To consider the higher cost of false positives, we constructed a cost function called weighted error rate. We used it as a fitness function in the PSO algorithm to minimize error rate and false positive rate.

3.1 Malware Detection using Heterogeneous Distance Function

Many classifiers require some measure of dissimilarity or distance between feature vectors, and their performance depends upon a good choice of the distance function. Especially the KNN classifier depends significantly on the metric used to compute distances between two feature vectors.

In this section, we propose a similarity metric for features extracted from PE file format. We used this metric to compute distances to find k nearest neighbors used in the KNN classifier. Note that the features used in our work are of various types and ranges. These

features can be divided into three types: numbers, bit fields, and strings. For example, number of sections or various addresses can be represented by numbers, section flags or characteristics by bit fields, and checksums by strings. Furthermore, some features have different ranges. For example, the number of sections is considerably smaller than the total number of called API functions. The proposed distance function can handle these types of features and also takes into account their different ranges.

3.1.1 Heterogeneous Distance Metric for PE Features

The most commonly used metric is the Euclidean distance which works well for numerical attributes (features). However, it does not appropriately handle nominal attributes. We proposed a suitable distance function for nominal and numeric attributes, as well as for boolean arrays.

Let $\mathbf{x}$ and $\mathbf{y}$ be two feature vectors of dimension m. The heterogenenous distance function is defined as follows:

$$\mathcal{D}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{3.1}$$

where

$$d_a(x, y) = \begin{cases} \mathcal{H}(x, y) & \text{if } a \text{ is a bit array} \\ \delta(x, y) & \text{if } a \text{ is a checksum} \\ \text{Norm\_diff}_a(x, y) & \text{if } a \text{ is a numeric} \\ \text{Norm\_vdm}_a(x, y) & \text{otherwise.} \end{cases} \tag{3.2}$$

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

$$\mathcal{H}(x, y) = |\{i | x_i \neq y_i, i = 1, \ldots, n\}| \tag{3.3}$$

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

$$\delta(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise.} \end{cases} \tag{3.4}$$

The Value Difference Metric (VDM) was introduced by [55] and the normalized VDM is defined as

$$\text{Norm\_vdm}_a(x, y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right| \tag{3.5}$$

where

- C is the number of classes,

- $n_{a,x,c}$ is the number of instances in 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:

$$\text{Norm\_diff}_a(x, y) = \frac{|x - y|}{4\sigma_a} \tag{3.6}$$

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) [56], and it can be used for our PE feature space since it handles both numeric and nominal attributes.

Since the distance functions d_a are metrics, $\mathcal{D}$ is also a metric. Properties of a metric, especially a triangle inequality, can be used to find nearest neighbors in a metric space more effectively.

Note that we distinguish between a checksum and a string attribute that is not a checksum. For the demonstration of distance function $\mathcal{D}$, we present the examples of PE attributes of each type:

- array of bits: Section flags, Characteristics, DllCharacteristics
- numeric attribute: number of sections, number of DLLs, size of all imports
- checksum: checksums of various pieces of the file content
- string: major/minor version of linker, operating system, subsystem

3.1.2 Malware Detection System

This section presents a system for detecting malware composed of a weighted KNN classifier and a statistical scoring technique introduced in [48]. We first describe the weighted KNN classifier and the statistical scoring technique. Then, we present our approach based on the combination of these classifiers and describe the architecture of the detection system.

3.1.2.1 The Weighted k-Nearest Neighbors Classifier

The k-Nearest Neighbors (KNN) classifier is one of the most popular supervised learning methods introduced by Fix and Hodges [57]. It is one of the simplest and best-known nonparametric algorithms in pattern classification.

Distance-weighted k-Nearest Neighbor procedure (WKNN) was first introduced in [58] as an improvement to KNN. This extension is based on the idea that closer neighbors are weighted more heavily than such neighbors that are far away from the query point. KNN implicitly assumes that all k nearest neighbors are equally important in making a classification decision, regardless of their distances to the query point. In WKNN, nearest

neighbors are weighted according to their distances to the query point as follows. Let $T = \{(x_1, c_1), \ldots, (x_m, c_m)\}$ be the training set, where x_i is training vector and c_i is the corresponding class label. For a query point x_q, its unknown class c_q is determined as follows. First, select the set $T' = \{(x_1, c_1), \ldots, (x_k, c_k)\}$ of k nearest neighbors to the query point x_q. Let $x_1, \ldots, x_k$ be k nearest neighbors of the query object and $d_1, \ldots, d_k$ the corresponding distances arranged in increasing order. The weight w_i for i-th nearest neighbor is defined as:

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1} & \text{if } d_k \neq d_1 \\ 1 & \text{otherwise} \end{cases} \tag{3.7}$$

The resulting class of the query point is then defined by the majority weighted vote as follows:

$$c_q = \arg\max_c \sum_{(x_i, c_i) \in T'} w_i \cdot \delta(c, c_i) \tag{3.8}$$

Note that finding the nearest neighbors is a very expensive operation for a dataset of the enormous size. The nearest neighbors can be found more efficiently by representing the training dataset as a tree.

3.1.2.2 Statistical Classifier – STATS

In this section, we present the scoring techniques that we used in our research. In contrast to the KNN classifier, a statistical-based classifier ignores the positions of points in our metric space and focuses on statistical properties of attribute (feature) values.

The following statistical classifier was introduced in [48]. Let $x = (x_1, \ldots, x_n)$ be a vector from our feature space and $\mathcal{M}$ a class of malware. Then probability

$$P(x \in \mathcal{M} | x_i = h) = \frac{n_{x_i,h,\mathcal{M}}}{n_{x_i,h}} \tag{3.9}$$

is the conditional probability that the output class of x is malware given that attribute x_i has the value h. Denote this probability by $p_i, i = 1, \ldots, n$. Note that the notations $n_{x_i,h,\mathcal{M}}$ and $n_{x_i,h}$ were used in the definition of VDM discussed in Section 3.1.1. Let function f with two parameters p_i and S_c be defined as

$$f(p_i, S_c) = \max\{0, p_i - S_c\}, \tag{3.10}$$

where S_c is an empirical constant. For each sample x we define a score as

$$\text{score} = \sum_{i=1}^{n} f(p_i, S_c) \tag{3.11}$$

From this score, we can determine a threshold S_s, above which we will classify a sample as malware. The decision rule is then defined as follows:

$$x \text{ is classified as } \begin{cases} \text{malware}, & \text{if score} > S_s, \\ \text{benign file}, & \text{otherwise.} \end{cases}$$

The pseudocode of the statistical-based classifier is described in Algorithm 3.1.

Algorithm 3.1 Statistical classifier – STATS

Input: original training set, query point x, distance metric $\mathcal{D}$
Output: label of x

```
score = 0
Compute probability vector (p_1, ..., p_n)
for i = 1 to n do
    if p_i > S_c then
        score += p_i − S_c
    end if
end for
if score > S_s then
    return malware
else
    return benign file
end if
```

In the rest of Section 3.1.2, the statistical-based classifier is denoted as STATS.

3.1.2.3 Our Approach

We propose a malware detection approach based on a combination of the well-known KNN classifier and the chosen statistically motivated classifier. In order to achieve higher detection rates, there should be some kind of diversity between the classifiers. KNN is a geometric-based classifier that uses labels of the nearest neighbors in some metric space to classify an unlabeled point. On the other hand, statistical-based approaches like Naive Bayes or the STATS classifier mentioned above use conditional probabilities of attributes of a sample point and do not use information about its position in feature space.

The proposed detection method works as follows. First, set the threshold to some sufficiently high value. Then compute the score using the chosen statistical scoring technique. If the score of the unknown file exceeds the threshold, then the resulting class will be malware. Otherwise, apply distance-weighted KNN. The pseudocode for the classification scheme is shown in Algorithm 3.2.

We chose KNN since it is a relatively accurate classifier for large datasets. The results of our experiments demonstrate that the statistical classifier can correctly classify samples lying in the area of feature space where the accuracy of KNN is low. The statistical classifier uses information from the training dataset in a different way than the KNN classifier. It checks whether a feature vector contains values typical for malware, in contrast to KNN that considers only differences between feature vectors.

Algorithm 3.2 Combination of the KNN and statistical classifier

Input: original training set, query point x, distance metric $\mathcal{D}$
Output: label of x

```
compute score from the statistical scoring technique
if score > threshold then
    return malware
else
    apply WKNN
end if
```

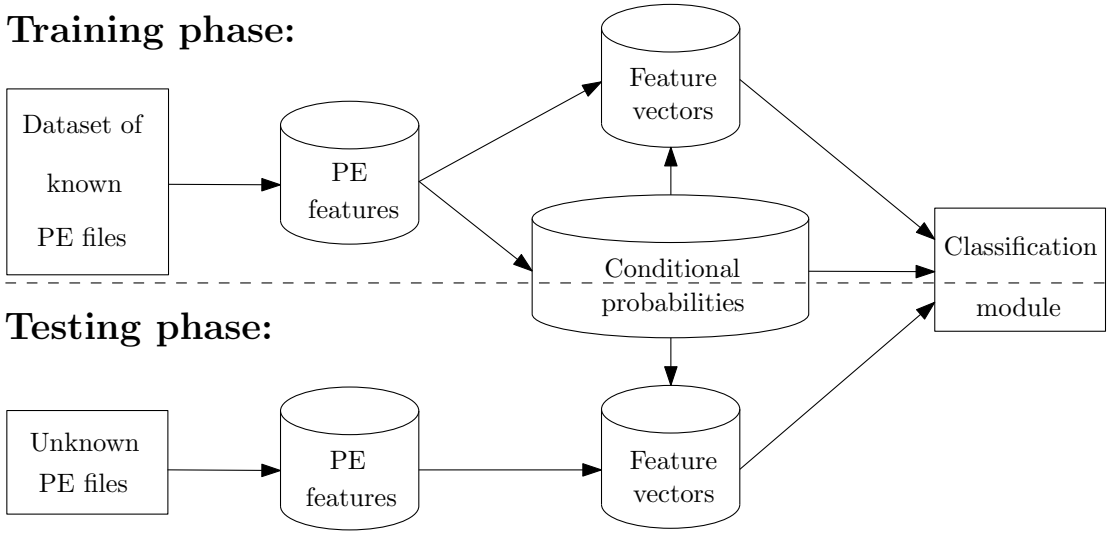


Figure 3.1: Architecture of the classification model

For example, consider a feature vector $x = (x_1, \ldots, x_n)$ containing only a few values (typically checksums), for which there is a high probability that x belongs to malware. Many other attributes could have previously unseen values or ones with a low prevalence. Therefore, malicious nearest neighbors could not be closer than benign nearest neighbors, and in this case, the KNN classifier would not be an appropriate method.

3.1.2.4 The system architecture

The system consists of three major components: a PE parser, a database of conditional probabilities, and a classification module, as illustrated in Figure 3.1.

The functionality of the PE parser is to extract all PE file header information, DLLs, and API functions from programs in the dataset and store all the extracted information in a database. Recall that our system is applied only to Windows PE files, and the PE parser extracts only the most useful features for discriminating between benign and malicious files. These features were determined by the feature selection algorithm Gain Ratio [59].

During the training phase, once the structural information of the PE files is extracted, the conditional probabilities $P(x \text{ is malware}|x_i = h)$ are computed for each PE attribute x_i and for each possible value h of attribute x_i. Note that only the PE features extracted from labeled samples of the training dataset are used in the computation of the conditional probabilities.

After extracting PE features and computing the conditional probabilities, feature vectors are created for every known PE file. The set of these feature vectors called training set will be used in the classification module, where the classification algorithm is applied to feature vectors of unknown PE files.

Experimental results and more details on heterogenenous distance function and the proposed ensemble classifier are described in the paper included in Section 4.1.

3.2 Distance Metric Learning using Particle Swarm Optimization

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. First, we present a modification of the heterogeneous distance function described in Section 3.1.1. We then present our proposed method based on a modification of the PSO algorithm used to learn the heterogeneous distance function. The paper included in Section 4.2 describes the malware detection system's architecture and presents the experimental results.

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. We modified the heterogeneous distance function described in Eq. (3.1) by considering weight for each feature. The weighted heterogeneous distance function is defined as follows:

$$\mathcal{D}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{a=1}^{m} w_a^2 d_a^2(x_a, y_a)} \tag{3.12}$$

The rest of this section presents our approach for finding the feature weights using Particle swarm optimization. Particle Swarm Optimization (PSO) [60] 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.

Concept and notation of the PSO elements concerning finding the feature weights used in weighted heterogeneous distance function in KNN classification is as follows:

- 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.

- 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 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. Note that in Section 3.5, we will also consider another optimization criterion focused on minimizing the false positive rate.

The PSO algorithm has three inputs: fitness function f, a training set T_{pso}, and vector p of feature importance scores [61] achieved from the feature selection algorithm. The pseudocode of the modified PSO algorithm is presented in Algorithm 3.3.

Algorithm 3.3 PSO algorithm

Input: fitness function f, T_{pso}, p
Output: vector of weights

initialize particles: $x_i = p \otimes Rand(0, \epsilon_1)$, $v_i = Rand(-\epsilon_2, \epsilon_2)$
repeat
  for each particle x_i **do**
    compute fitness function $f(x_i)$
    if $f(x_i) > pbest_i$ **then**
      $pbest_i = f(x_i)$
      $p_i = x_i$
    end if
  end for
  select the most successful particle in swarm so far, and denote it by p_g
  for each particle x_i **do**
    $v_i = \omega v_i + \text{Rand}(0, \phi_1) \otimes (p_i - x_i) + \text{Rand}(0, \phi_2) \otimes (p_g - x_i)$
    $x_i = x_i + v_i$
  end for
until maximum number of iterations is attained
return global best position

Rand$(0, \epsilon)$ represents a vector of random numbers uniformly distributed in $[0, \epsilon]$, where ϵ is a small constant. Operation $\otimes$ denotes 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 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 global search while lower values tend to local search. Parameters ϕ_1 and ϕ_2

represent the weights, and they 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.

Inertia weight ω depends on the number of iterations. At the first iteration, ω is set to one, and it linearly decreases at each iteration to the value $\omega_{min} = 0.8$.

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 stuck into the local minima.

3.3 Malware Detection using LSTM

This section presents the transformation of PE features using the LSTM network [62]. As a result, classification results achieved using transformed feature space improved significantly. We experimented with various LSTM architectures, which we used for feature transformation. The paper included in Section 4.3 describes experimental results and presents our approach in more detail.

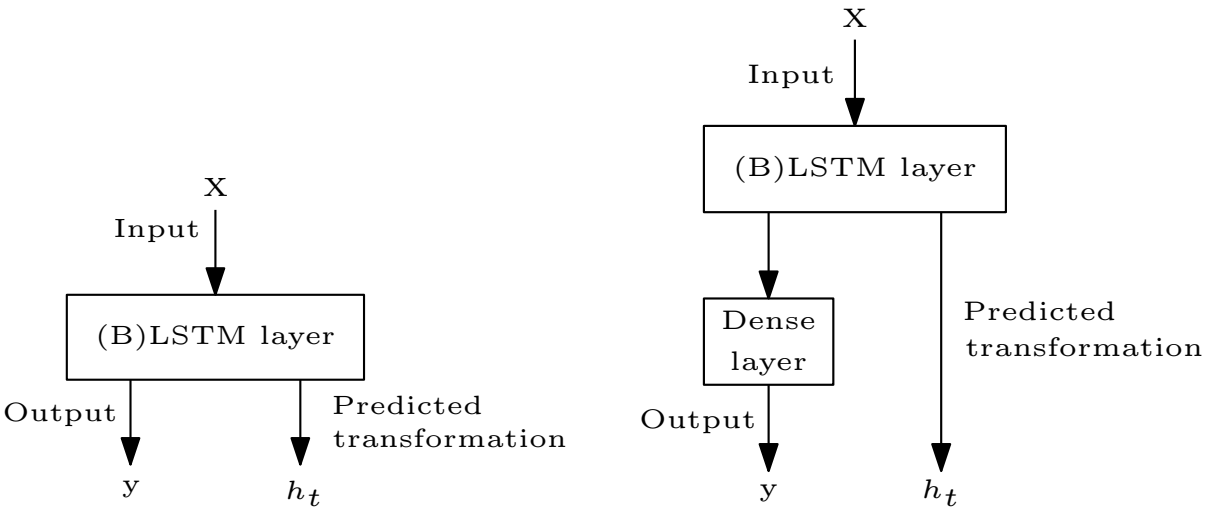
Our research is not limited to only LSTM networks; however, a bidirectional version of LSTM networks (BLSTM) [63] was also included in our experiments. We considered two different types of neural networks: the *Basic version* consisting of one (B)LSTM layer and the *Deep version* with four (B)LSTM layers, each layer containing 50 LSTM units equal to the number of input features. All networks were trained up to 50 epochs with a batch size of 32, Adam optimization, and mean squared error loss function.

3.3.1 Type 1

The first type of LSTM network we experimented with is based on autoencoder's architecture [64]. In this case, we worked only with explanatory variables with a network designed to predict the same values which were given on input. The predicted transformation was taken from the penultimate layer's last hidden state. Schema of the Type 1 transformer is illustrated in Figure 3.2a. Note that hidden state h_t represents the short-term memory, and the detailed description of LSTM network is presented in paper included in Section 4.3.

3.3.2 Type 2

The second type was similar to the regular use of the LSTM network, where we work with both the explanatory and response variables. For prediction, we used the last hidden state of the penultimate LSTM layer as with Type 1. The last layer was occupied by a single



(a) Schema of Basic version Type 1 transformer.

(b) Schema of Basic version Type 2 transformer.

Figure 3.2: Two different types of transformers.

neuron with a sigmoid activation function. Diagram of the Type 2 transformer is presented in Figure 3.2b.

Experiment workflow consits of the following steps:

1. **Feature extraction**. Python module `pefile` [65] was used to extract features from the PE file format.

2. **Feature preprocessing**. Numeric features were transformed min-max normalization, strings were transformed into term frequency times inverse document frequency representation using `TfidfVectorizer` from the `scikit-learn` library [66], and non-string data were encoded into numeric values using feature hashing `FeatureHasher` also from the `scikit-learn`.

3. **Feature selection**. Using Principal Component Analysis algorithm, we reduced the dimensionality by extracting the most relevant features.

4. **Feature transformation**. Features were transformed using (B)LSTM network as described above.

5. **Classification**. Several state-of-the-art ML algorithms were applied on the transformed dataset.

6. **Evaluation**. Classification results were evaluated using common evaluation metrics.

Experimental results and a detailed description of our approach can be found in the paper included in Section 4.3.

3.4 Malware Family Classification using DML

This section concerns the application of selected state-of-the-art distance metric learning techniques to the multiclass classification problem for six prevalent malware families and benign files. This multiclass classification problem is more challenging than a binary classification problem where the goal is to distinguish between malicious and benign files.

3.4.1 Distance Metric Learning

This section provides basic information on distance metric learning. 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 follows:

$$d_w(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i^2 \left(x_i - y_i\right)^2} \tag{3.13}$$

The goal of learning the weighted Euclidean distance is to find an appropriate weight vector $w = (w_1, \ldots, w_n)$ concerning some optimization criterion, usually minimizing error rate. Several other distance functions have been presented [56]. In order to improve results, many weighting schemes were proposed. A review of feature weighting methods for lazy learning algorithms was proposed in [67].

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

$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^\top \mathbf{M} (\mathbf{x} - \mathbf{y})} \ , \tag{3.14}$$

where $\mathbf{M}$ is a positive semidefinite matrix. Mahalanobis distance can be considered as a generalization of Euclidean distance, since if $\mathbf{M}$ is the identity matrix, then $d_{\mathbf{M}}$ in Eq. (3.14) is reduced to common Euclidean distance. If $\mathbf{M}$ is diagonal, then this corresponds to learning the feature weights $M_{ii} = w_i$ defined for weighted Euclidean distance in Eq. (3.13).

The goal of learning the Mahalanobis distance is to find an appropriate matrix $\mathbf{M}$ concerning some optimization criterion. Regarding the KNN classifier, the goal can be defined as to find a matrix $\mathbf{M}$ which is estimated from the training set, leading to the lowest error rate of the KNN classifier. Since a positive semidefinite matrix $\mathbf{M}$ can always be decomposed as $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, distance metric learning problem can be viewed as finding either $\mathbf{M}$ or $\mathbf{L} = \mathbf{M}^{\frac{1}{2}}$. Mahalanobis distance defined in Eq. (3.14) expressed in terms of the matrix $\mathbf{L}$ is defined as

$$d_{\mathbf{M}}(\mathbf{x}, \mathbf{y}) = d_{\mathbf{L}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{L}^\top (\mathbf{x} - \mathbf{y})\|_2 \tag{3.15}$$

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

$$\mathbf{x}' = \mathbf{L}\mathbf{x} \tag{3.16}$$

Mahalanobis distance of two points $\mathbf{x}, \mathbf{y}$ from the original space defined in Eq. (3.14) corresponds to the Euclidean distance between transformed points $\mathbf{x'} = \mathbf{Lx}, \mathbf{y'} = \mathbf{Ly}$ defined as follows:

$$d_{\mathbf{L}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{L}^\top(\mathbf{x} - \mathbf{y})\|_2 = \sqrt{(\mathbf{x'} - \mathbf{y'})^\top (\mathbf{x'} - \mathbf{y'})} \tag{3.17}$$

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

Distance metric learning has attracted a lot of attention in the machine learning field and is still an active research area [68].

3.4.2 Our Approach

The paper included in Section 4.4 presents the application of three state-of-the-art distance metric learning techniques to malware families classification.

We employed the following three DML techniques, Large Margin Nearest Neighbor (LMNN) [69], Neighborhood Component Analysis (NCA) [70], and Metric Learning for Kernel Regression (MLKR) [71], to learn the Mahalanobis distance metric to improve multiclass classification performance for our dataset containing six prevalent malware families and benign files. LMNN and NCA were selected since they were designed to improve the KNN clas- sifier. MLKR was included in our experiments since our preliminary experiments indicated the potential to achieve competitive results. All three DML techniques are described in the paper included in Section 4.4.

Our approach can be divided into the following steps:

1. **Feature extraction**. The features used in our experiments were extracted from the PE file format via Python module `pefile` [65]. We extracted 358 numeric features that are based on static information only.

2. **Feature normalization**. All features were normalized using procedure `preprocessing.normalize` from the Scikit-learn library.

3. **Feature normalization**. We then employed the six feature selection methods also imported from the Scikit-learn library. The lowest error rate was achieved for 25 selected features by Recursive Feature Elimination [72] with Logistic Regression estimator.

4. **Distance Metric Learning**. We employed the DML algorithm to learn the Mahalanobis distance metric.

5. **Feature Vector Transformation** We transformed original feature set by linear transformation described in Eq. (3.16).

6. **Classification**. We applied several state-of-the-art machine learning algorithms on transformed feature vectors.

7. **Evaluation**. We evaluated classification results using common performance metrics.

Experimental results are described in the paper included in Section 4.4.

3.5 Malware Detection using DML

Similar to Section 3.4, this section deals with the application of DML to malware detection. However, this section focuses on binary classification concerning the following 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.

In this section, we first describe the architecture of the malware detection system using DML. Then, we present our approach for dealing with the task (2). More details on our approach and the experimental results can be found in the paper included in Section 4.5.

3.5.0.1 Architecture

We present the architecture of 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 3.3 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 testing set. The training set is used for training a distance metric and classifier.

Step 2: **Parsing binaries.** For each sample, we extract and store information from PE file format. These features will be preprocessed in the following two steps, and relevant features only 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 appeared 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.** Feature selection algorithm is used to determine the relevant features and produced the final version of the feature set.

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

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

Steps 7: **Evaluation:** Performance metrics are used to measure classification results.

Computing the conditional probabilities for nominal features and performing feature selection algorithm for all three types of features are done only to the training samples. The

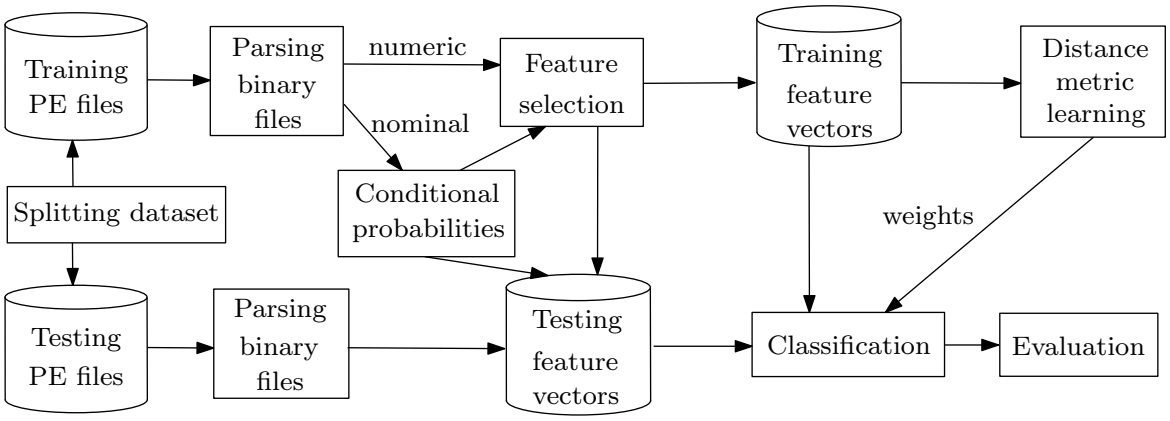


Figure 3.3: Architecture of our proposed malware detection model using distance metric learning.

corresponding conditional probabilities and selected features are applied to design training and testing feature vectors.

3.5.1 Minimizing of False Positive Rate

To deal with task (2), we proposed two different approaches. The first approach is based on the PSO algorithm with the optimization criterion called WERR based on a weighted error rate to penalize false positives. The second approach is based on the modification of LMNN.

3.5.1.1 WERR

In this section, we propose optimization criteria for detecting as much malware as possible while keeping a low false positive rate. To consider different costs of a false positive and false negative, we adjust the loss function that penalized 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 of false positives (FP) and number of false negatives (FN) as

$$\mathrm{ERR} = \frac{\mathrm{FP} + \mathrm{FN}}{|T_{test}|}, \tag{3.18}$$

where $|T_{test}|$ is the number of testing samples. We modify Eq. (3.18) 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 false positive:

$$\mathrm{WERR} = \frac{c\,\mathrm{FP} + \mathrm{FN}}{|T| + (c-1)\mathrm{FP}} \tag{3.19}$$

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

$$c\ \mathrm{FP}_{new} - \mathrm{FP}_{old} < \mathrm{FN}_{old} - \mathrm{FN}_{new} \tag{3.20}$$

One point of view on the WERR criterion is that we agree to "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 described in Algorithm 3.3. Experimental results are presented in the paper included in Section 4.5.

3.5.1.2 Modification of LMNN

This section first briefly presents LMNN and then introduces our modification of LMNN suitable for the task (2).

Large Margin Nearest Neighbor (LMNN) [69] is one of the state-of-the-art distance metric learning algorithms used to learn a Mahalanobis distance metric for KNN classification. LMNN consists of two steps. In the first step, for each instance, $\mathbf{x}$, a set of k nearest instances belonging to the same class as $\mathbf{x}$ (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 $\mathbf{x}$ than instances from different classes separated by a large margin. The Mahalanobis distance metric is estimated by solving semidefinite programming problem defined as:

$$\min_{\mathbf{L}} \sum_{i,j:j\rightsquigarrow i} \Big(d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)^2 + \\ + \mu \sum_{k:y_i \neq y_k} [1 + d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)^2 - d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_k)^2]_+ \Big) \tag{3.21}$$

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

Finally, $[x]_+$ is defined as the hinge-loss, i.e. $[x]_+ = \max\{0, x\}$.

Regarding minimizing FPR using LMNN, we modified Eq. (3.21) 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 denotes the set of k target neighbors of x_i. Then the modification of LMNN concerning minimizing false positives is as follows:

$$\min_{\mathbf{L}} \sum_{x_i \in T} \sum_{x_j \in N_i} \Big(d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)^2 + \\ + \mu \sum_{x_k \in T : y_i \neq y_k} \eta_k [1 + d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j)^2 - d_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_k)^2]_+ \Big) \tag{3.22}$$

where η_k denotes cost of false positive and it is define as follows:

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

Similar 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 the WERR criterion is that the modification of LMNN takes into account the distance between a sample and its impostor.

The paper included in Section 4.5 contains more details on the approach and presents experimental resuts.

Chapter 4

Author's Relevant Papers

This section includes a collection of five papers that present the main results of this thesis. A brief description is provided for each paper.

The following papers are presented:

Paper 1 [A.1] *Malware Detection Using a Heterogeneous Distance Function.*. This paper presents the distance function that can handle various types of PE features. The distance function is used in the proposed malware detection system based on the combination of the k-Nearest Neighbors classifier and the statistical-based classifier.

Paper 2 [A.2] *Distance Metric Learning using Particle Swarm Optimization to Improve Static Malware Detection.* The distance from the Paper 1 is modified by considering the feature weights. The PSO algorithm is applied to the problem of finding the most appropriate feature weights.

Paper 3 [A.3] *Representation of PE Files using LSTM Networks.* Various LSTM network architectures were used to transform the PE features, and state-of-the-art ML algorithms were applied to the transformed features to improve classification results.

Paper 4 [A.4] *Improving Classification of Malware Families using Learning a Distance Metric.* This paper describes the application of three DML algorithms to learn the Mahalanobis distance metric to improve multiclass classification performance for the dataset containing six prevalent malware families and benign files.

Paper 5 [A.5] *Application of Distance Metric Learning to Automated Malware Detection.* This paper extends Paper 2 and partially Paper 4 and presents the architecture of the malware detection model based on distance metric learning. Besides minimizing the error rate, the paper also focuses on the task that considers the cost of false positives.

Fig. 4.1 illustrates the relationships among all five author's relevant papers.

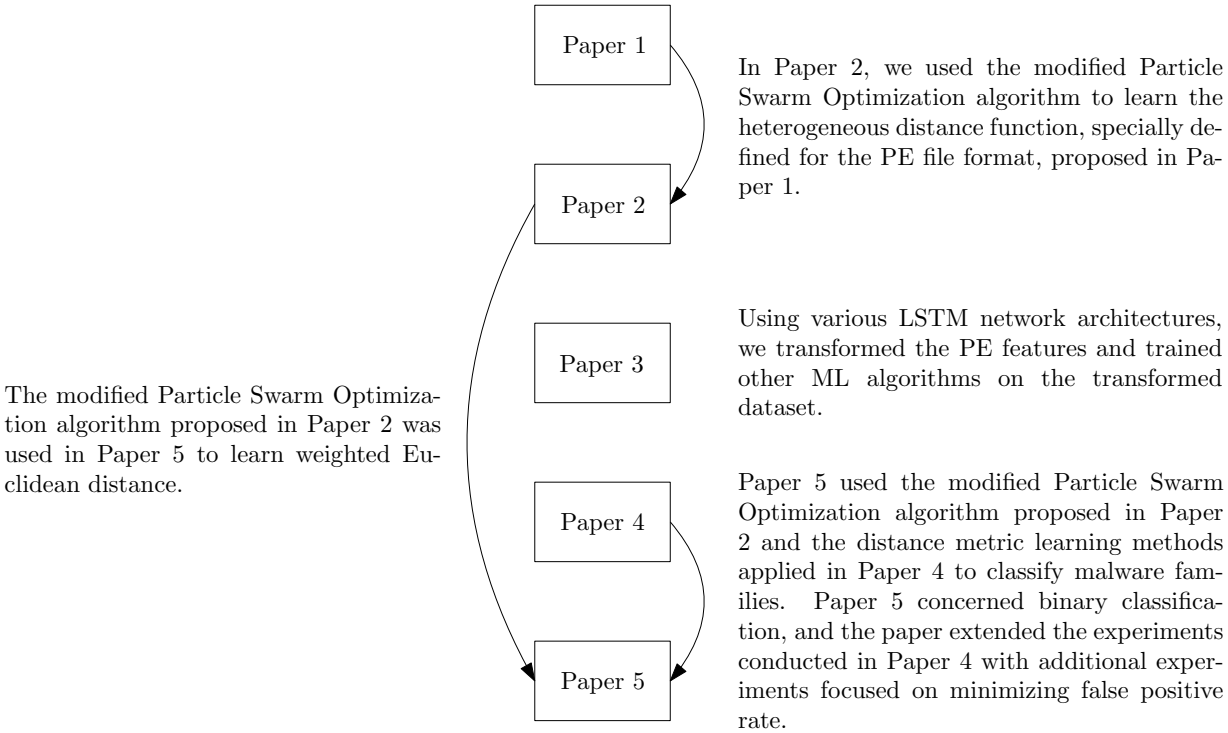


Figure 4.1: Relationships among author's relevant papers.

4.1 Paper 1 - Malware Detection Using a Heterogeneous Distance Function

Mgr. Martin Jureček (60%), prof. Ing. Róbert Lórencz, CSc. (40%)
In: *Computing and Informatics*. Volume 37, No. 3, pp. 759-780, 2018.

The most commonly used metric is the Euclidean distance which works well for numerical attributes. However, it does not appropriately handle nominal attributes, which are included in PE file format. This paper describes the distance function that can handle various types of PE features, such as nominal and numeric features and boolean array.

The paper also presents a malware detection approach based on a combination of the well-known KNN classifier and the statistical classifier. The statistical classifier uses information from the training dataset in a different way than the KNN classifier. It checks whether a feature vector contains values typical for malware, in contrast to KNN that considers only differences between feature vectors.

The paper also deals with clustering malware into families. Partitioning Around Medoids algorithms using the heterogeneous distance function was applied for five prevalent malware families.

Computing and Informatics, Vol. 37, 2018, 759–780, doi: 10.4149/cai_2018_3_759

MALWARE DETECTION USING A HETEROGENEOUS DISTANCE FUNCTION

Martin Jureček, Róbert Lórencz

Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague, Czech Republic
e-mail: {martin.jurecek, lorencz}@fit.cvut.cz

Abstract. Classification of automatically generated malware is an active research area. The amount of new malware is growing exponentially and since manual investigation is not possible, automated malware classification is necessary. In this paper, we present a static malware detection system for the detection of unknown malicious programs which is based on combination of the weighted k-nearest neighbors classifier and the statistical scoring technique from [12]. We have extracted the most relevant features from portable executable (PE) file format using gain ratio and have designed a heterogeneous distance function that can handle both linear and nominal features. Our proposed detection method was evaluated on a dataset with tens of thousands of malicious and benign samples and the experimental results show that the accuracy of our classifier is 98.80 %. In addition, preliminary results indicate that the proposed similarity metric on our feature space could be used for clustering malware into families.

Keywords: Malware detection system, feature selection, similarity measure, k-nearest neighbors classifier, partitioning around medoids

1 INTRODUCTION

The problem of automated malware detection presents challenges for antivirus vendors (AV). Most AV rely primarily on a signature detection technique which is relatively simple and efficient rule-based method for detecting known malware [10]. Signature (unique hex code strings) of the malware is extracted and added to the database. The antivirus engine compares the contents of a file with all malware

signatures in its database and if a match is found, the file is reported as malware. A good signature must capture malware with a minimal false positive probability.

The major weakness of signature detection is its inability to detect obfuscated and zero-day malware. A number of non-signature based malware detection techniques have been proposed [19, 11, 20]. These techniques are used in an effort to detect new or unknown malware and can be grouped into two main approaches: static and dynamic heuristic methods. Static methods can be based on an analysis of the file format without actually running the program. Dynamic analysis aims to examine a program which is executed in a real or virtual environment. Non-signature based malware detection techniques suffer from two main problems: high false positive rate and large processing overhead.

In this paper, we present a static malware detection system based on combination of the statistical classifier and the k-nearest neighbors (KNN) classifier. Experimental results indicate that the combination of the classifiers may provide a potential benefit for detecting samples not detected by KNN.

In our work we propose the following four main contributions:

- We present a feature space extracted from PE file format by using feature selection method based on information gain.
- We design a new distance function that can handle both nominal and linear attributes.
- We present a malware detection system for detecting previously unknown malicious PE files. In order to achieve a higher detection rate, the system uses a combination of two different kinds of classifiers.
- We evaluate the effectiveness of our detection system and distance function on a real-world malware collection.

The rest of the paper is organized in the following way. Section 2 provides an overview of previous work on malware classification. In Section 3 we present the feature space and the distance function used in KNN classifier. Section 4 discusses our proposed detection technique, while Section 5 covers our experimental results. Finally, conclusions are given in Section 6.

2 RELATED WORK

In this section, we survey some relevant previous work in the area of classification schemes for malware detection. To maintain the focus, we mainly discuss the work using static detection based on machine learning techniques. Then we briefly discuss various existing statistical-based scores and also several methods that rely on dynamic analysis.

Schultz et al. [19] introduced the concept of data mining for detecting previously unknown malware. In their research they presented three different static feature sources for malware classification: information from the portable executable

(PE) header and strings and byte sequences extracted from binaries. These features were used in three different kinds of algorithms: an inductive rule-based learner, a probabilistic method, and a multi-classifier system. A rule induction algorithm called Ripper [4] was applied to find patterns in the dynamic-link library (DLL) data (such as the list of DLLs used by the binary, the list of DLL function calls, and the number of different system calls used within each DLL). The well-known probabilistic method, learning algorithm Naive Bayes, was used to find patterns in the string data and n-grams of byte sequences. Multinomial Naive Bayes algorithm that combined the output of several classifiers reached the highest detection rate of 97.76 %. The authors tested the data mining methods against standard signatures and their results indicate that the data mining detection rate of a previously unknown malware was twice as high in comparison to the signature-based methods.

Kolter and Maloof [11] improved the Schulz's third technique by using overlapping byte sequences instead of non-overlapping sequences. They used different kinds of classifiers: naive Bayes, instance-based learner, similarity-based classifier called TFIDF, Support Vector Machine (SVM), Decision Trees (DT) and boosted variants of SVM, DT and TFIDF. Authors evaluated their classifiers performance by computing the area under a receiver operating characteristic curve. Boosted Decision tree model (J48) achieved the best accuracy, an area under the ROC curve of 0.996 and outperformed the rest of the classifiers.

In other studies, operational code (opcode) has been used as static information for malware detection. Common techniques are based on the frequency of appearance of opcode-sequences [18], examination of opcode frequency distribution difference between malicious and benign code [2], or identification of critical instruction sequences [22]. Other techniques use similarity of executables based on opcode graphs [17]. However, some executable files cannot be disassembled properly, therefore the opcode approach is not always feasible [2].

The more recent work [23] contains three statistical-based scoring techniques, namely Hidden Markov models, Simple substitution distance, and Opcode graph-based detection. Authors showed that a combination of these scoring techniques with a Support Vector Machine yields significantly more robust results than those obtained using any of the individual scores.

We also briefly mention a few existing detection methods that rely on dynamic analysis. Examples of the information we can obtain from dynamic analysis include application programming interface (API) and system calls, instruction traces, memory writes, registry changes, and so on. In [24], an artificial neural network was employed to detect previously unknown worms based on the computer's behavioral measures. Eskandari et al. [25] extracted a set of program API calls and combined them with control flow graph. Qiao et al. [26] proposed a new malware analysis method based on frequency analysis of API call sequences. Note that dynamic analysis is time-consuming as each malware sample must be executed for a certain time period.

3 FEATURE SPACE AND METRIC

We design our proposed detection system for the portable executable (PE) file format [5], which is the most widely used file format for malware samples. In order to classify an executable file in the PE format, we extract static format information and translate it into a feature vector suitable for classification.

3.1 Feature Space

Before presenting attributes used in our feature vector, let us firstly look at the general outline of the PE file format. A simplified overview of the PE file format is illustrated in Figure 1.

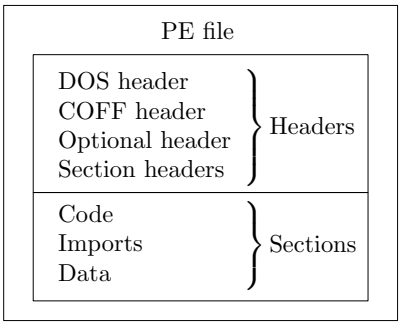


Figure 1. PE file structure

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 followed immediately by the section headers which provide information about sections, including locations, sizes, and characteristics. Sections divide the file content into code, resources, and various types of data.

Based on our empirical studies and analysis of the PE format, we selected a set of static features that are helpful in distinguishing malware and benign files and used gain ratio for selection the most relevant features.

3.1.1 Features Selection

In order to determine which attribute in a given training set is the most useful for discriminating between the classes, we use entropy-based measure, information gain (IG) [13]. The information gain is the expected reduction in entropy caused by knowing the value of attribute a. $\mathrm{IG}(\mathcal{T}, a)$ of an attribute a relative to training

dataset $\mathcal{T}$ and is defined as

$$\mathrm{IG}(\mathcal{T}, a) = \mathrm{Entropy}(\mathcal{T}) - \sum_{v \in V(a)} \frac{|\mathcal{T}_v|}{|\mathcal{T}|} \mathrm{Entropy}(\mathcal{T}_v) \tag{1}$$

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. Note that the entropy of the training dataset $\mathcal{T}$ is given by:

$$\mathrm{Entropy}(\mathcal{T}) = -\sum_{c \in C} p_c \log_2 p_c \tag{2}$$

where p_c is the proportion of $\mathcal{T}$ belonging to class c.

The information gain measure is biased towards attributes with many values. One way of avoiding this difficulty is to use a modification of the measure called the gain ratio (GR) [15]. The gain ratio measure penalizes attributes with large numbers of possible values by incorporating a term called split information (SI):

$$\mathrm{SI}(\mathcal{T}, a) = -\sum_{i=1}^{d} \frac{|\mathcal{T}_i|}{|\mathcal{T}|} \log_2 \frac{|\mathcal{T}_i|}{|\mathcal{T}|} \tag{3}$$

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 $\mathrm{SI}(\mathcal{T}, a)$ is the entropy of $\mathcal{T}$ with respect to the values of attribute a. The gain ratio is then defined as

$$\mathrm{GR}(\mathcal{T}, a) = \frac{\mathrm{IG}(\mathcal{T}, a)}{\mathrm{SI}(\mathcal{T}, a)} \tag{4}$$

and we select only features with the highest values of gain ratio.

3.1.2 Our Proposed Feature Space

The following feature set was extracted using gain ratio and used in our work:

- Many fields from the PE headers, such as the number of sections, date/time stamp, major or minor versions of linker, operating system, image, subsystem; sizes and addresses of data directories; DLL characteristics, etc. Table 1 lists all features that are derived from the PE headers. For detailed description of these features, see Chapter 3 in [5].
- Features from sections and their headers: VirtualSize, VirtualAddress, SizeOfRawData, PointerToRawData, Section Flags (see Chapter 4 in [5]), and features not contained within the PE structure, including entropies and checksums of sections.
- Resources of the PE file are used to provide supporting content, such as icons, fonts, strings and other elements. In case of malicious files, resources are often

used to store code and configuration data. For example, the number of resources and the number of types of resources were used in our work.

- Overlay is a data that is appended at the end of the executable file. We considered the size of the overlay.
- Other features: the size of all imports, the number of DLLs referred, the number of APIs referred.

Feature	*Feature*
NumberOfSections	MajorOperatingSystemVersion
TimeDateStamp	MajorImageVersion
SizeOfOptionalHeader	MajorSubsystemVersion
Characteristics	MinorSubsystemVersion
MajorLinkerVersion	SizeOfImage
MinorLinkerVersion	CheckSum
AddressOfEntryPoint	Subsystem
ImageBase	DllCharacteristics
SectionAlignment	NumberOfRvaAndSizes
FileAlignment	Addresses and sizes of data directories

Table 1. List of features from the PE headers

Note that our feature set is similar to that in the existing works [12, 21, 1].

3.2 Distance Function

Many classifiers require some measure of dissimilarity or distance between feature vectors, and its performance depends upon a good choice of distance function. Especially the KNN classifier depends significantly on the metric used to compute distances between two feature vectors.

In this section, we propose a similarity metric on our feature space. We used this metric to compute distances to find k nearest neighbors used in KNN classifier. Note that the features used in our work are of various types and sizes. These features can be divided into three types: numbers, bit fields, and strings. For example, number of sections or various addresses can be represented by numbers, section flags or characteristics by bit fields, and checksums by strings. Furthermore, some features have different ranges. For example, the number of sections is considerably smaller than the total number of called API functions. The proposed distance function can handle these types of features and also takes into account their different ranges.

The most commonly used metric is the Euclidean distance which works well for numerical attributes. However, it does not appropriately handle nominal attributes. The Value Difference Metric (VDM) [27] was proposed to define a suitable distance function for nominal attributes. A version of the VDM without a weighting scheme

is defined for values x and y of an attribute a as:

$$\mathrm{VDM}_a(x,y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right|^q \tag{5}$$

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.

Since the Euclidean distance is not suitable for nominal attributes, and VDM is inappropriate for numeric attributes, heterogeneous metric can be used to handle our feature space. Wilson and Martinez introduced Heterogeneous Value Difference Metric (HVDM) [29] which is defined for feature vectors $\mathbf{x}$ and $\mathbf{y}$ as:

$$\mathrm{HVDM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{6}$$

where m is the number of attributes of the feature vector and the definition of distance function $d_a(x,y)$ depends on the type of attribute a as follows:

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown,} \\ \mathrm{NORM\_VDM}_a(x,y), & \text{if } a \text{ is nominal,} \\ \mathrm{NORM\_DIFF}_a(x,y), & \text{if } a \text{ is linear.} \end{cases} \tag{7}$$

Functions $\mathrm{NORM\_VDM}_a(x,y)$ and $\mathrm{NORM\_DIFF}_a(x,y)$ are defined as:

$$\mathrm{NORM\_VDM}_a(x,y) = \sum_{c=1}^{C} \left| \frac{n_{a,x,c}}{n_{a,x}} - \frac{n_{a,y,c}}{n_{a,y}} \right|, \tag{8}$$

$$\mathrm{NORM\_DIFF}_a(x,y) = \frac{|x-y|}{4\sigma_a} \tag{9}$$

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

3.2.1 Our Proposed Distance Function

Since the feature vector described in Section 3.1.2 contains more types of nominal attributes we propose the following distance function:

$$\mathcal{D}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{10}$$

where

$$d_a(x,y) = \begin{cases} \mathcal{H}(x,y), & \text{if } a \text{ is an array of bits,} \\ \delta(x,y), & \text{if } a \text{ is a checksum,} \\ \text{NORM\_DIFF}_a(x,y), & \text{if } a \text{ is numeric,} \\ \text{NORM\_VDM}_a(x,y), & \text{otherwise } (a \text{ is a string}). \end{cases} \tag{11}$$

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

$$\mathcal{H}(x,y) = |\{i \mid x_i \neq y_i, i = 1, \ldots, n\}| \tag{12}$$

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

$$\delta(x,y) = \begin{cases} 0, & \text{if x = y,} \\ 1, & \text{otherwise.} \end{cases} \tag{13}$$

Since the distance functions d_a are metrics, $\mathcal{D}$ is also a metric. Properties of a metric, especially the triangle inequality, can be used in effective finding of nearest neighbors in a metric space.

Note that we distinguish between a checksum and a string attribute that is not a checksum. For the demonstration of distance function $\mathcal{D}$ on our feature space, we present the examples of attributes of each type:

- array of bits: Section flags, Characteristics, DllCharacteristics,
- numeric attribute: number of sections, number of DLLs, size of all imports,
- checksum: checksums of various pieces of the file content,
- string: major/minor version of linker, operating system, subsystem.

4 PROPOSED SYSTEM FOR DETECTING MALWARE

In this section, we present a system for detecting malware which is composed of a KNN classifier and a statistical scoring technique.

4.1 The k-Nearest Neighbors Classifier

The k-nearest neighbors (KNN) classifier is one of the most popular supervised learning methods introduced by Fix and Hodges [8]. It is one of the simplest and best-known nonparametric algorithms in pattern classification.

Let $T = \{(x_1, c_1), \ldots, (x_m, c_m)\}$ be the training set, where x_i is training vector and c_i is the corresponding class label. Given a query point x_q, its unknown class c_q is determined as follows. First, select the set $T' = \{(x_1, c_1), \ldots, (x_k, c_k)\}$ of k nearest

neighbors to the query point x_q. Then assign the class label to the query point x_q by majority vote of its nearest neighbors:

$$c_q = \arg\max_c \sum_{(x_i,c_i)\in T'} \delta(c, c_i) \tag{14}$$

where c is a class label, c_i is the class label for i^{th} neighbor among k nearest neighbors of the query point, and $\delta(c, c_i)$ takes a value of one if $c = c_i$ and zero otherwise. Cover and Hart [6] found that if the number of samples approaches infinity, the nearest-neighbor error rate is bounded from above by twice the Bayes error rate.

Distance-weighted k-nearest neighbor procedure (WKNN) was first introduced in [7] as an improvement to KNN. This extension is based on the idea that closer neighbors are weighted more heavily than such neighbors that are far away from the query point. KNN implicitly assumes that all k nearest neighbors are equally important in making a classification decision, regardless of their distances to the query point. In WKNN, nearest neighbors are weighted according to their distances to the query point as follows. Let $x_1, \ldots, x_k$ be k nearest neighbors of the query object and $d_1, \ldots, d_k$ the corresponding distances arranged in increasing order. The weight w_i for i-th nearest neighbor is defined as:

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1}, & \text{if } d_k \neq d_1, \\ 1, & \text{otherwise.} \end{cases} \tag{15}$$

The resulting class of the query point is then defined by the majority weighted vote as follows:

$$c_q = \arg\max_c \sum_{(x_i,c_i)\in T'} w_i \cdot \delta(c, c_i). \tag{16}$$

Note that finding the nearest neighbors is a very expensive operation due to the enormous size of our dataset. The nearest neighbors can be found more efficiently by representing the training dataset as a tree.

4.2 The Statistical-Based Classifiers

In this section, we present the scoring techniques that we used in our research. In the case of the statistical-based classifier, we ignore the positions of points in our metric space and we focus on statistical properties of attribute values, in contrast to the KNN classifier.

4.2.1 Naive Bayes

This section introduces the Naive Bayes classifier [28] for binary (two-class) classification problems. A Naive Bayes classifier is a probabilistic algorithm based on Bayes' Theorem that predicts the class with the highest *a posteriori* probability. Assume a set of two classes $\{\mathcal{C}, \mathcal{M}\}$, where $\mathcal{C}$ denotes the class of benign samples

and $\mathcal{M}$ denotes the class of malware. Training datasets are provided and a new (unknown) sample, which is represented by a feature vector $x = (x_1, \ldots, x_n)$, is presented. Let $P(\mathcal{M}|x)$ denote the probability that a sample is malicious given the feature vector x that describes the sample. Similarly, $P(\mathcal{C}|x)$ denotes the probability that a sample is benign given the feature vector x that represents the sample. The Naive Bayes classification rule is stated as

$$\begin{aligned} &\text{If } P(\mathcal{M}|x) < P(\mathcal{C}|x), x \text{ is classified as benign sample,} \\ &\text{If } P(\mathcal{M}|x) > P(\mathcal{C}|x), x \text{ is classified as malware.} \end{aligned} \tag{17}$$

The *a posteriori* probabilities $P(C|x)$ may be expressed in terms of the *a priori* probabilities and the $P(x|C)$ probabilities using Bayes' theorem as

$$P(C|x) = \frac{P(x|C)\ P(C)}{P(x)}. \tag{18}$$

Assuming that the values of the attributes (features) are conditionally independent on one another, Equation (18) may be expressed as

$$P(C|x) = \frac{\prod_{i=1}^{n} P(x_i|C)\ P(C)}{P(x)}. \tag{19}$$

Probabilities $P(x_i|C)$ can be estimated from the training set by counting the attribute values for each class. More precisely, the probability $P(x_i = h|C)$ is represented as the number of samples of class C in the training set having the value h for attribute x_i, divided by the number of samples of class C in the training set. The output of the classifier is the highest probability class C':

$$C' = \arg\max_{C} \left(P(C)\ \prod_{i=1}^{n} P(x_i|C) \right). \tag{20}$$

4.2.2 Statistical Classifier – STATS

The following statistical classifier was introduced in [12]. Let $x = (x_1, \ldots, x_n)$ be a vector from our feature space and $\mathcal{M}$ a class of malware. Then probability

$$P(x \in \mathcal{M}|x_i = h) = \frac{n_{x_i,h,\mathcal{M}}}{n_{x_i,h}} \tag{21}$$

is the conditional probability that the output class of x is malware given that attribute x_i has the value h. Denote this probability by p_i, $i = 1, \ldots, n$. Note that the notations $n_{x_i,h,\mathcal{M}}$ and $n_{x_i,h}$ were used in the definition of VDM discussed in Section 3.2. Define a function f with two parameters p_i and S_c as

$$f(p_i, S_c) = \max\{0, p_i - S_c\} \tag{22}$$

where S_c is an empirical constant. For each file x we define a score as

$$\text{score} = \sum_{i=1}^{n} f(p_i, S_c). \tag{23}$$

From this score, we can determine a threshold S_s, above which we will classify a file as malware. The decision rule is then defined as follows:

$$x \text{ is classified as } \begin{cases} \text{malware}, & \text{if score} > S_s, \\ \text{benign file}, & \text{otherwise.} \end{cases} \tag{24}$$

The pseudocode of the statistical-based classifier is described in Algorithm 1.

Algorithm 1 Statistical classifier – STATS

Input: original training set, query point x, distance metric $\mathcal{D}$
Output: label of x

```
score = 0
Compute probability vector (p_1, ..., p_n)
for i = 1 to n do
    if p_i > S_c then
        score += p_i − S_c
    end if
end for
if score > S_s then
    return malware
else
    return benign file
end if
```

In the rest of this paper, the statistical-based classifier is denoted as STATS.

4.3 Our Approach

We propose a malware detection approach based on a combination of the well-known KNN classifier and the chosen statistical motivated classifier. In order to achieve higher detection rates, there should be some kind of diversity between the classifiers. KNN is a geometric-based classifier which uses labels of the nearest neighbors in some metric space to classify an unlabeled point. On the other hand, statistical-based approaches like Naive Bayes or the STATS classifier mentioned above use conditional probabilities of attributes of sample point and do not use information about its position in feature space.

The proposed detection method works as follows. First, set the threshold to some sufficiently high value. Then compute score using the chosen statistical scoring technique. If the score of the unknown file exceeds the threshold, then the resulting

class will be malware, otherwise apply distance-weighted KNN. The pseudocode for the classification scheme is shown in Algorithm 2.

Algorithm 2 Our detection system

Input: original training set, query point x, distance metric $\mathcal{D}$
Output: label of x

```
compute score from the statistical scoring technique
if score > threshold then
    return malware
else
    apply WKNN
end if
```

The reason why we chose KNN is that it is a relatively accurate classifier for large datasets and the results of our experiments demonstrate that the statistical classifier is able to correctly classify samples lying in the area of feature space where the accuracy of KNN is low. The statistical classifier uses information from the training dataset in a different way than the KNN classifier. It checks whether a feature vector contains values typical for malware, in contrast to KNN that considers only differences between feature vectors.

For example, consider a feature vector $x = (x_1, \ldots, x_n)$ containing only a few values (typically checksums), for which there is a high probability that x belongs to malware. Many other attributes could have previously unseen values or ones with a low prevalence. Therefore, malicious nearest neighbors could not be closer than benign nearest neighbors and in this case, KNN classifier would not be an appropriate method.

4.3.1 The System Architecture

The system consists of three major components: a PE parser, a database of conditional probabilities and a classification module, as illustrated in Figure 2.

The functionality of the PE parser is to extract all PE format file's header information, DLLs, and API functions from programs in the dataset and store all the extracted information in a database. Recall that our system is applied only to Windows PE files and the PE parser extracts only the most useful features for discriminating between benign and malicious files. These features were determined by the feature selection algorithm mentioned in Section 3.

During the training phase, once the structural information of the PE files is extracted, the conditional probabilities $P(x \text{ is malware}|x_i = h)$ are computed for each PE attribute x_i and for each possible value h of attribute x_i. Note that only the PE features extracted from labeled samples of the training dataset are used in the computation of the conditional probabilities.

After extracting PE features and computing the conditional probabilities, feature vectors are created for every known PE file. The set of these feature vectors

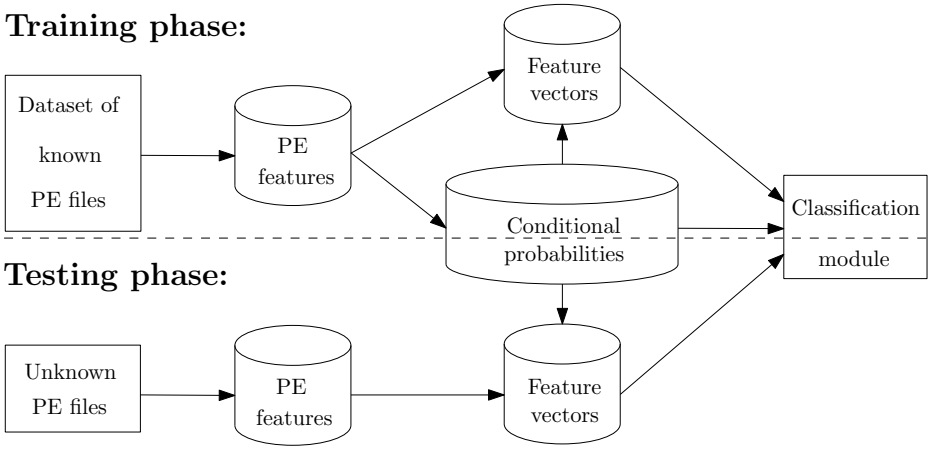


Figure 2. Architecture of the classification model

called training set will be used in the classification module where the classification algorithm is applied to feature vectors of unknown PE files.

5 EVALUATION RESULTS AND ANALYSIS

In this section, we introduce the performance metrics and present the results of our experiments. We compare our approach with several other machine learning methods for malware detection.

5.1 Performance Metrics

We present the evaluation 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):

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{25}$$

It is defined on a given test set as the percentage of correctly classified instances. However, since our dataset is not well-balanced, the accuracy measure could be an inappropriate measure of performance. If we use a classifier which labels every sample as benign, then TN will be very high and TP will be very low. As a result, the accuracy obtained on our dataset will be very high.

The second parameter, True Positive Rate (TPR) (or detection rate), is defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{26}$$

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

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}}. \tag{27}$$

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

We also evaluate our classifier using Receiver Operating Characteristic (ROC) analysis [3]. ROC curve is represented as a two-dimensional plot, in which true positive rate is plotted against false positive rate at various threshold settings. The area under the ROC curve (AUC) serves as the performance measure of our detection techniques. An AUC of 1.0 represents the ideal case where both false positive and false negative equal zero. On the other hand, AUC of 0.5 means that the classifier's performance is no better than flipping a coin.

5.2 Dataset

The dataset used in this research consists of a total of 101,604 Windows programs in the PE file format, out of which 21,087 are malicious and 80,517 are legitimate or benign programs. There were no duplicate programs in our dataset. The malicious and benign programs were obtained from the laboratory of the industrial partner.

In order to expose any biases in the data, we used the 5-fold cross-validation procedure. Generally in k-fold cross-validation [14], the dataset is randomly divided into k subsets of equal size, where k-1 subsets are used for training and 1 subset is used for testing. In each of the k folds a different subset is reserved for testing and the accuracies obtained for each fold are averaged to produce a single cross validation estimate.

In the cluster analysis we used five prevalent malware families that have appeared during the year 2016. Specifically, we have used the following malware families:

- Allaple – a polymorphic network worm that spreads to other computers and performs denial-of-service (DoS) attacks.

- Dinwod – a trojan horse that silently downloads and installs other malware on the compromised computer.
- Virlock – a ransomware that locks victims' computer and demands a payment in order to unlock it.
- Virut – a virus with backdoor functionality that operates over an IRC-based communications protocol.
- Vundo – a trojan horse that displays pop-up advertisements and also injects JavaScript into HTML pages.

5.3 Classification Results

We implemented the classifiers as described in Section 4. The feature space and the distance function proposed in Section 3 were used in the KNN and the WKNN classifiers. The combination of the WKNN and the statistical scoring technique from [12] is denoted as WKNN_STATS and the combination of the WKNN and the Naive Bayes classifier is denoted as WKNN_NB. For each experiment, we performed 5-fold cross-validation that gives approximately unbiased estimate of a classifier's accuracy.

In our first experiment, we attempt to distinguish between benign and malicious PE files. We used accuracy, discussed in Section 5.1, as a comparison criterion for comparing classifiers. The accuracies obtained after applying the WKNN and the KNN classifiers for various numbers of nearest neighbors are depicted in Figure 3.

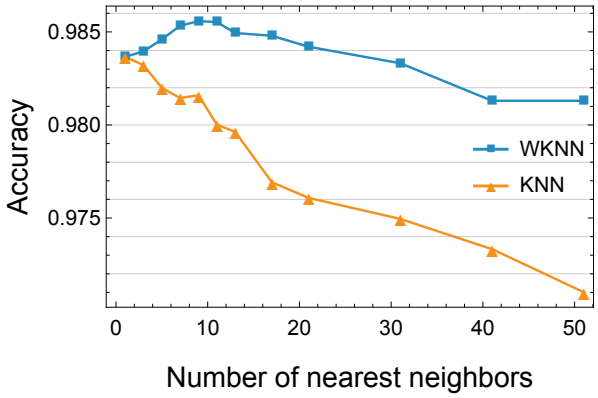


Figure 3. Classification accuracies of the WKNN and the KNN classifiers, for various numbers of nearest neighbors

The WKNN classifier achieved the highest accuracy using nine nearest neighbors, while the KNN classifier achieved the highest accuracy using only three nearest neighbors.

The classification results of the classifiers implemented in this research are listed in Table 2.

Classifier	*TPR*	*FPR*	*Accuracy*
KNN	96.23 %	1.07 %	98.37 %
WKNN	96.82 %	0.99 %	98.56 %
NB	82.78 %	1.17 %	95.50 %
STATS	90.08 %	**0.76** %	97.34 %
WKNN_NB	97.37 %	1.05 %	98.62 %
WKNN_STATS	**98.08** %	1.01 %	**98.80** %

Table 2. Classification results of six approaches implemented in this work

Among these classifiers, the WKNN_STATS outperformed others with the highest accuracy of 98.8 %. Note that the WKNN_STATS classifier was tested for various threshold values, and the best result was achieved with the following parameters:

- the number of nearest neighbors $k = 9$ used in the WKNN classifier,
- the thresholds $S_c = 0.8$ and $S_s = 0.53$ used in the STATS classifier.

In addition to that, we constructed the ROC curves which are shown in Figure 4 for three chosen classifiers.

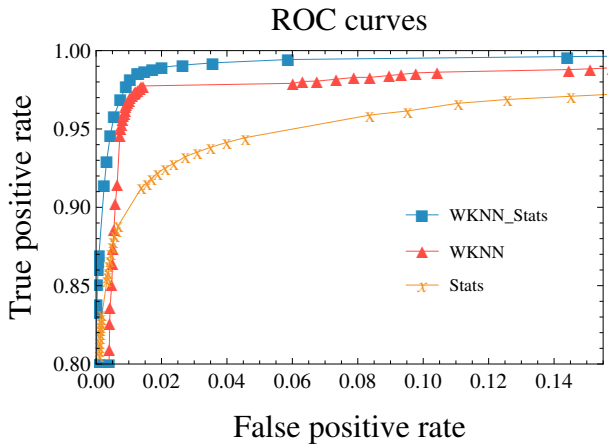


Figure 4. ROC curves for the WKNN, STATS and WKNN_STATS classifiers

We can conclude from Figure 4 that a combination of the classifiers outperforms both individual classifiers.

Table 3 reports the AUC for three classifiers discussed in Section 4 and two related static methods: KM [11] and PE Miner [21]. As the table illustrates, WKNN_STATS classifier provides the best AUC value with 0.998. The ROC curve and AUC values confirm that our experiment provides excellent results regarding malware detection.

Classifier	*AUC*
WKNN	0.993
STATS	0.983
WKNN_STATS	**0.998**
KM	0.996
PE Miner	0.992

Table 3. Comparison of the AUC value for five static methods

5.4 Clustering Results

In the second experiment we apply cluster analysis to five prevalent malware families described in Section 5.2. First, we present the clustering algorithm used in this experiment and then describe the evaluation measures and show the results.

5.4.1 Partitioning Around Medoids

Partitioning around medoids (PAM) proposed by Kaufman and Rousseeuw [9] is a well-known technique for performing non-hierarchical clustering. The reason why we have decided to use the PAM algorithm is that it allows clustering with respect to any distance metric. The pseudocode of the PAM algorithm is described in Algorithm 3.

Algorithm 3 PAM algorithm

Input: Number of clusters k, set of data points T
Output: k clusters

```
1: Initialize: randomly select k data points from T to become the medoids
2: Assign each data point to its closest medoid
3: for all cluster do
4:    identify the observation that would yield the lowest average distance if it were
      to be re-assigned as the medoid
5:    if the observation is not current medoid then
6:       make this observation the new medoid
7:    end if
8: end for
9: if at least one medoid has changed then
10:   go to step 2
11: else
12:   end the algorithm.
13: end if
```

5.4.2 Evaluation Measures

We evaluated the quality of clusters through the measures of purity and silhouette coefficient (SC). Let n_{ij} be the number of samples of class i in cluster C_j and let $p_{ij} = \frac{n_{ij}}{|C_j|}$. The probability p_{ij} is the probability that a randomly selected sample from cluster C_j belongs to class i. The purity of cluster C_j is defined as $\text{Purity}(C_j) = \max_i p_{ij}$.

The overall purity value is defined as the weighted sum of individual purities for each cluster, taking into account the size of each cluster:

$$\text{Purity} = \frac{1}{n} \sum_{j=1}^{k} |C_j| \text{Purity}(C_j). \tag{28}$$

To measure the quality of clusters, we compute the average silhouette coefficient [16] for each cluster. Suppose there are n samples $x_1, \ldots, x_n$ that have been divided into k clusters $C_1, \ldots, C_k$. Consider a sample $x_i \in C_j$, and define the average distance between x_i to all other samples in cluster C_j:

$$a(x_i) = \frac{1}{|C_j| - 1} \sum_{\substack{y \in C_j \\ y \neq x_i}} d(x_i, y). \tag{29}$$

Let $b_k(x_i)$ be the average distance from sample $x_i \in C_j$ to all samples in cluster C_k not containing x_i:

$$b_k(x_i) = \frac{1}{|C_k|} \sum_{y \in C_k} d(x_i, y). \tag{30}$$

After computing $b_k(x_i)$ for all clusters C_k, where $k \neq j$, we select the minimum of those numbers:

$$b(x_i) = \min_{k \neq j} b_k(x_i). \tag{31}$$

The silhouette coefficient of x_i is obtained by combining $a(x_i)$ and $b(x_i)$ as follows:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}. \tag{32}$$

The value of $s(x_i)$ in Equation (32) can vary between -1 and 1. It is desirable to have the value $s(x_i)$ as close to 1 as possible, since then the clusters are well-separated. The average silhouette coefficient for a given cluster is defined as the average value of $s(x_i)$ over all samples in the cluster.

5.4.3 Experimental Results

We computed the silhouette coefficient, as discussed above. For computing the silhouette coefficient, we used our proposed distance function on the feature space

Majority Class	*Size*	*Purity*	*SC*
Allaple	424	0.9343	0.3298
Dinwod	285	0.7429	0.7172
Virlock	452	0.9771	0.5635
Virut	337	0.68	0.2389
Vundo	252	0.6886	0.1921
Overall	1 750	0.8298	0.3883

Table 4. The purity and the silhouette coefficient for clusters

discussed in Section 3. Table 4 summarizes the results of silhouette coefficient based experiments using the PAM algorithm.

According to the experiences of authors of SC [16], silhouette coefficient values between 0.7 and 1.0 indicate excellent clustering results. SC values between 0.5 and 0.7 indicate a reasonable structure of cluster. SC values below 0.25 indicate that no substantial structure has been found.

Regarding the clustering malware into families, our results show that the quality of clusters varies widely, depending on the particular family. From the results in Table 4, we see that the PAM algorithm can correctly classify the malware family with an accuracy of about 68 % to over 97 %, depending on the particular family.

Note that such accuracies are lower than those obtained with classifiers presented in Section 4. The reason is that distinguishing between malware families is a more challenging problem than a binary classification of malware and benign files.

6 CONCLUSION

In this paper, we proposed a new detection system using a combination of the k-nearest neighbors classifier and the statistical-based classifier. The system can automatically detect unknown malware samples. The feature set used in our work was a collection of properties extracted from the PE file format. We designed a new distance function that is capable of handling various types of features.

Experimental results indicate that the combination of the classifiers may provide a potential benefit to detect samples not detected by KNN. We compared the different classification methods and concluded that the combination of the weighted k-nearest neighbors classifier and the statistical-based classifier achieves the highest accuracy, 98.8 %. The results also indicate that the proposed heterogeneous distance function and the feature space are appropriate for malware detection and could be also used for clustering malware into families.

The proposed static malware detection system is relatively easy to implement, and can be utilized to support commercial antivirus systems. For future work, it would be interesting to experiment with additional statistical scoring techniques in the context of malware classification.

Acknowledgements

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

REFERENCES

[1] Asquith, M.: Extremely Scalable Storage and Clustering of Malware Metadata. Journal of Computer Virology and Hacking Techniques, Vol. 12, 2016, No. 2, pp. 49–58, doi: 10.1007/s11416-015-0241-3.

[2] Bilar, D.: Opcodes as Predictor for Malware. International Journal of Electronic Security and Digital Forensics, Vol. 1, 2007, No. 2, pp. 156–168, doi: 10.1504/IJESDF.2007.016865.

[3] Bradley, A. P.: The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. Pattern Recognition, Vol. 30, 1997, No. 7, pp. 1145–1159, doi: 10.1016/S0031-3203(96)00142-2.

[4] Cohen, W. W.: Learning Trees and Rules with Set-Valued Features. Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI/IAAI), Vol. 1, 1996, pp. 709–716.

[5] Microsoft Corporation: Visual Studio, Microsoft Portable Executable and Common Object File Format Specification, Revision 9.3, 2015.

[6] Cover, T.—Hart, P.: Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory, Vol. 13, 1967, No. 1, pp. 21–27, doi: 10.1109/TIT.1967.1053964.

[7] Dudani, S. A.: The Distance-Weighted k-Nearest-Neighbor Rule. IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-6, 1976, No. 4, pp. 325–327, doi: 10.1109/TSMC.1976.5408784.

[8] Fix, E.—Hodges Jr., J. L.: Discriminatory Analysis – Nonparametric Discrimination: Consistency Properties. Technical Report, DTIC Document, 1951.

[9] Kaufman, L.—Rousseeuw, P. J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, Wiley Series in Probability and Statistics, Vol. 334, 2009.

[10] Kephart, J. O.—Arnold, W. C.: Automatic Extraction of Computer Virus Signatures. 4th Virus Bulletin International Conference, 1994, pp. 178–194.

[11] Kolter, J. Z.—Maloof, M. A.: Learning to Detect and Classify Malicious Executables in the Wild. The Journal of Machine Learning Research, Vol. 7, 2006, pp. 2721–2744.

[12] Merkel, R.—Hoppe, T.—Kraetzer, C.—Dittmann, J.: Statistical Detection of Malicious PE-Executables for Fast Offline Analysis. In: De Decker, B., Schaumüller-Bichl, I. (Eds.): Communications and Multimedia Security (CMS 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6109, 2010, pp. 93–105.

[13] Mitchell, T. M.: Machine Learning. New York, 1997.

[14] Picard, R. R.—Cook, R. D.: Cross-Validation of Regression Models. Journal of the American Statistical Association, Vol. 79, 1984, No. 387, pp. 575–583, doi: 10.1080/01621459.1984.10478083.

[15] Quinlan, J. R.: Induction of Decision Trees. Machine Learning, Vol. 1, 1986, No. 1, pp. 81–106, doi: 10.1007/BF00116251.

[16] Rousseeuw, P. J.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Journal of Computational and Applied Mathematics, Vol. 20, 1987, pp. 53–65, doi: 10.1016/0377-0427(87)90125-7.

[17] Runwal, N.—Low, R. M.—Stamp, M.: Opcode Graph Similarity and Metamorphic Detection. Journal in Computer Virology, Vol. 8, 2012, No. 1–2, pp. 37–52, doi: 10.1007/s11416-012-0160-5.

[18] Santos, I.—Brezo, F.—Nieves, J.—Penya, Y. K.—Sanz, B.—Laorden, C.—Bringas, P. G.: Idea: Opcode-Sequence-Based Malware Detection. In: Massacci, F., Wallach, D., Zannone, N. (Eds.): Engineering Secure Software and Systems (ESSoS 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5965, 2010, pp. 35–43.

[19] Schultz, M. G.—Eskin, E.—Zadok, F.—Stolfo, S. J.: Data Mining Methods for Detection of New Malicious Executables. Proceedings of the 2001 IEEE Symposium on Security and Privacy (S & P 2001), IEEE Computer Society, 2001, pp. 38–49, doi: 10.1109/SECPRI.2001.924286.

[20] Shabtai, A.—Moskovitch, R.—Elovici, Y.—Glezer, C.: Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey. Information Security Technical Report, Vol. 14, 2009, No. 1, pp. 16–29, doi: 10.1016/j.istr.2009.03.003.

[21] Shafiq, M. Z.—Tabish, S. M.—Mirza, F.—Farooq, M.: PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In: Kirda, E., Jha, S., Balzarotti, D. (Eds.): Recent Advances in Intrusion Detection (RAID 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5758, 2009, pp. 121–141.

[22] Siddiqui, M.—Wang, M. C.—Lee, J.: Data Mining Methods for Malware Detection Using Instruction Sequences. Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications (AIA '08), 2008, pp. 358–363.

[23] Singh, T.—Di Troia, F.—Corrado, V. A.—Austin, T. H.—Stamp, M.: Support Vector Machines and Malware Detection. Journal of Computer Virology and Hacking Techniques, Vol. 12, 2016, No. 4, pp. 203–212.

[24] Stopel, D.—Boger, Z.—Moskovitch, R.—Shahar, Y.—Elovici, Y.: Application of Artificial Neural Networks Techniques to Computer Worm Detection. Proceedings of the 2006 IEEE International Joint Conference on Neural Networks (IJCNN '06), 2006, pp. 2362–2369.

[25] Eskandari, M.—Hashemi, S.: A Graph Mining Approach for Detecting Unknown Malwares. Journal of Visual Languages and Computing, Vol. 23, 2012, No. 3, pp. 154–162, doi: 10.1016/j.jvlc.2012.02.002.

[26] Qiao, Y.—Yang, Y.—Ji, L.—He, J.: Analyzing Malware by Abstracting the Frequent Itemsets in API Call Sequences. 2013 12th IEEE International Conference

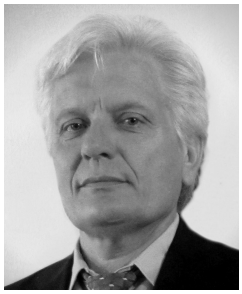
on Trust, Security and Privacy in Computing and Communications (TrustCom), 2013, pp. 265–270.

[27] Stanfill, C.—Waltz, D.: Toward Memory-Based Reasoning. Communications of the ACM, Vol. 29, 1986, No. 12, pp. 1213–1228, doi: 10.1145/7902.7906.

[28] Webb, A. R.—Copsey, K. D.: Statistical Pattern Recognition. Third Edition. Wiley, 2011.

[29] Wilson, D. R.—Martinez, T. R.: Improved Heterogeneous Distance Functions. Journal of Artificial Intelligence Research, Vol. 6, 1997, No. 1, pp. 1–34.

Martin Jureček graduated from the Charles University in Prague, Faculty of Mathematics and Physics, with the specialization in mathematical methods of information security. He is now a Ph.D. student at the Faculty of Information Technology of the Czech Technical University in Prague. His main research interests focus on the application of machine learning and artificial intelligence approaches to malware detection. Another area of his interest is cryptography and information security.



Róbert Lórencz graduated from the Faculty of Electrical Engineering of the Czech Technical University in Prague in 1981. He received his Ph.D. degree in 1990 from the Institute of Measurement and Measuring Methods, Slovak Academy of Sciences in Bratislava. Currently he is Full Professor at the Faculty of Information Technology of the Czech Technical University in Prague. His research interests are cryptography and arithmetic units for cryptography primitives, various cryptoanalysis methods of block and stream ciphers. Another topic of his interest is alternative arithmetic for numerical computation.

4.2 Paper 2 - Distance Metric Learning using Particle Swarm Optimization to Improve Static Malware Detection

Mgr. Martin Jureček (70%), prof. Ing. Róbert Lórencz, CSc. (30%)
In *Proceedings of 6th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 725-732, Malta, Valletta, 2020

This paper deals with finding the most appropriate parameters of the heterogenenous distance metric proposed in Paper 1. This task is formulated as the following optimization problem: to minimize the error rate of the KNN classifier using a weighted heterogeneous distance function. A modification of a biologically-motivated algorithm, Particle Swarm Optimization (PSO), was used to handle this problem. In the proposed modification of PSO, results from the feature selection algorithm (feature importance scores) are used to initialize the particles instead of random initialization. The purpose of the initialization is to accelerate PSO, i.e., reducing the searching space is done using the feature selection algorithm results.

The main purpose of this research is to improve the accuracy of the malware detection system based on the KNN classifier in the case when standard methods such as feature selection or algorithm tuning have already been applied.

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
{*jurecmar, lorencz*}*@fit.cvut.cz*

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.

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

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

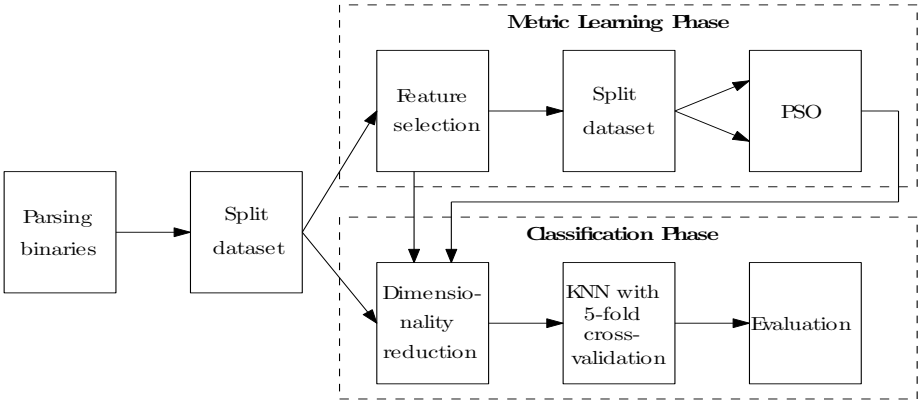


Figure 1: Architecture of our proposed malware detection system.

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.

4.3 Paper 3 - Representation of PE Files using LSTM Networks

Mgr. Martin Jureček (60%), Bc. Matouš Kozák (40%)
In *Proceedings of 7th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 516-525, Virtual event, 2021

This paper presents the transformation of PE features using various LSTM network architectures. The research was not limited to only LSTM networks, however, a bidirectional version of LSTM networks (BLSTM) was also included in our experiments. State-of-the-art ML classifiers were then trained on the transformed feature vectors.

On these transformed datasets, we ran a cross-validation benchmark using multiple supervised ML algorithms to see whether the feature transformation based on (B)LSTM networks can increase the performance of the ML algorithms in comparison to the performance achieved for the non-transformed dataset.

The main idea, the use of output sequence from the LSTM as an input to ML classifier, was the contribution of the author of this dissertation thesis. This idea was successfully developed by Matouš Kozák as a bachelor thesis [73] supervised by the author of this dissertation thesis.

Representation of PE Files using LSTM Networks

Martin Jureček and Matouš Kozák
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
{*jurecmar, kozakmat*}*@fit.cvut.cz*

Keywords: Malware Detection, PE File Format, Recurrent Neural Network, Long Short-term Memory.

Abstract: An ever-growing number of malicious attacks on IT infrastructures calls for new and efficient methods of protection. In this paper, we focus on malware detection using the Long Short-Term Memory (LSTM) as a preprocessing tool to increase the classification accuracy of machine learning algorithms. To represent the malicious and benign programs, we used features extracted from files in the PE file format. We created a large dataset on which we performed common feature preparation and feature selection techniques. With the help of various LSTM and Bidirectional LSTM (BLSTM) network architectures, we further transformed the collected features and trained other supervised ML algorithms on both transformed and vanilla datasets. Transformation by deep (4 hidden layers) versions of LSTM and BLSTM networks performed well and decreased the error rate of several state-of-the-art machine learning algorithms significantly. For each machine learning algorithm considered in our experiments, the LSTM-based transformation of the feature space results in decreasing the corresponding error rate by more than 58.60 %, in comparison when the feature space was not transformed using LSTM network.

1 INTRODUCTION

Malware is a software that conducts malicious activities on the infected computer. Cybersecurity professionals across the globe are trying to tackle this unwanted behaviour. Even though they are developing defense systems on a daily basis, cybercriminals process at the same, if not, in a faster manner.

Antivirus programs detect more than 370,000 malicious programs each day (AV-test, 2019), and the number keeps rising. Although Windows remains the most attacked platform, macOS and IoT devices are becoming attractive targets as well. The most popular weapon for cybercriminals on Windows remains Trojan, for instance, Emotet, WannaCry, Mirai and many others (Symantec, 2019).

In May 2017, the world was struck by new ransomware WannaCry. This virus quickly spread all around the world, infecting more than 230,000 computers in 150 countries. Between infected organizations were, e.g. FedEx, O2, or Britain's NHS and the cost of damage was estimated at around 4 billion dollars (Latto, 2020).

In the paper, we focus on static malware detection where features are collected from the PE file format. We are not examining files' working behaviour for multiple reasons. Firstly, extracting API calls from executable files needs to be performed in a sandbox environment to secure the leak of possible malicious activities into our system. However, this is bypassed by the unnatural behaviour of many programs in these surroundings. Secondly, it's time-consuming running large datasets and capturing their activities.

During the last years, the current trend is to used malware detection framework 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.

This paper aims to explore whether the LSTM networks can transform features to more convenient feature space, and as a result, improve the classification accuracy. This problem is tackled in two stages. In the first stage, we collect malware and benign files, extract useful information, prepare and select the best features to create our dataset. The second stage consists of training different LSTM network architectures, transforming our dataset using these networks, and evaluating our results with the help of several supervised machine learning (ML) algorithms.

The structure of the paper is as follows. In Section 2, we review related work on malware detection using neural networks, especially recurrent neural nets.

Section 3 describes fundamental background, such as PE file format and LSTM networks. In Section 4, we describe feature preprocessing and propose feature transformation using LSTM networks. Description of our experimental setup, from the dataset and hardware used to final evaluation using supervised ML algorithms, is placed in Section 5. We conclude our work in Section 6.

2 RELATED WORK

In this part, we review related research in the field of static malware detection. We focused on the papers linked to neural networks, notably recurrent neural networks (RNNs). However, we didn't find much work dealing with the use of LSTM networks as a feature pre-treatment before the classification itself.

In (Lu, 2019), the authors used opcodes (operation code, part of machine language instruction (Barron, 1978)) extracted from a disassembled binary file. From these opcodes, they created a language with the help of word embedding. The language is then processed by the LSTM network to get the prediction. They achieved an AUC-ROC score of 0.99, however, their dataset consisted of only 1,092 samples.

A much larger dataset of 90,000 samples was used in (Zhou, 2018). They used an LSTM network to process API call sequences combined with the convolutional neural network to detect malicious files. While also using static and dynamic features, they managed to achieve an accuracy of 97.3%.

Deep neural networks were also used in (Saxe and Berlin, 2015) with the help of Bayesian statistics. They worked with a large dataset of more than 400 thousand binaries. With fixed FPR at 0.1%, they reported AUC-ROC of 0.99964 with TPR of 95.2%.

The authors of (Hardy et al., 2016) used stacked autoencoders for malware classification and achieved an accuracy of 95.64% on 50,000 samples.

In (Vinayakumar et al., 2018), they trained the stacked LSTM network and achieved an accuracy of 97.5% with an AUC-ROC score of 0.998. That said they focused on android files and collected only 558 APKs (Android application package).

3 BACKGROUND

In this chapter, we explain the necessary background for this paper. The first part deals with the Portable Executable file format, describing the use cases and structure. In the second part, we study the LSTM networks in detail. In the end, we also briefly mention the autoencoder networks.

3.1 Portable Executable

Portable Executable (**PE**) **format** is a file format for Windows operation systems (Windows NT) executables, DLLs (dynamic link libraries) and other programs. Portable in the title denotes the transferability between 32-bit and 64-bit systems. The file format contains all basic information for the OS loader (Kowalczyk, 2018).

The structure of the PE file is strictly set as follows. Starting with MS-DOS stub and header, followed with file, optional, and section headers and finished with program sections as illustrated in Figure 1. The detailed description can be found in (Karl Bridge, 2019).

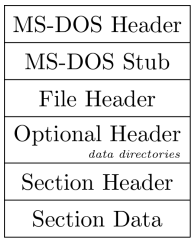


Figure 1: Structure of a PE file.

3.2 LSTM Network

Long short-term memory or shortly **LSTM** network is a subdivision of recurrent neural networks. This network architecture was introduced in (Hochreiter and Schmidhuber, 1997). The improvement lies in replacing a simple node from RNN with a compound unit consisting of *hidden state* or h_t (as with RNNs) and so-called *cell state* or c_t. Further, adding *input node* g_t compiling the input for every time step t and three *gates* controlling the flow of information. Gates are binary vectors, where 1 allows data to pass through, 0 blocks the circulation. Operations with gates are handled by using Hadamard (element-wise) product $\odot$ with another vector (Leskovec et al., 2020).

As mentioned above, the LSTM cell is formed by a group of simple units. The key difference from RNN is the addition of three **gates** which regulate the input/output of the cell.

Note that W_x, W_h and $\vec{b}$ with subscripts in all of the equations below are learned weights matrices and vectors respectively, and f denotes an activation function, e.g. sigmoid. Subscripts are used to distinguish matrices and vectors used in specific equations.

1. **Input Gate.** Determines which information can be allowed inside the unit:

$$i_t = f(W_{x_i} x_t + W_{h_i} h_{t-1} + \vec{b}_i) \tag{1}$$

2. **Forget Gate.** Allows us to discard information from memory we do not longer need:

$$f_t = f(W_{x_f} x_t + W_{h_f} h_{t-1} + \vec{b}_f) \tag{2}$$

3. **Output Gate.** This gate learns what data is paramount at a given moment and enables the unit to focus on it:

$$o_t = f(W_{x_o} x_t + W_{h_o} h_{t-1} + \vec{b}_o) \tag{3}$$

The **input node** takes as an input x_t and previous hidden state:

$$g_t = f(W_{x_g} x_t + W_{h_g} h_{t-1} + \vec{b}_g) \tag{4}$$

As an activation function is typically used *tanh* even though *ReLU* might be easier to train (Lipton et al., 2015).

The **cell state** is calculated as follows:

$$c_t = i_t \odot g_t + f_t \odot c_{t-1} \tag{5}$$

In equation (5), we can see the intuition behind using the input and forget gates. The gates handle how much of the input node and previous cell state we allow into the cell. This formula is the essential improvement to simple RNNs as the forget gate vector applied to the previous cell state is what allows the gradient to safely pass during backpropagation, thus abolishing the problem of *vanishing gradient* (Leskovec et al., 2020).

The **hidden state** is then updated with the content of the current cell state modified with output gate o_t as follows:

$$h_t = f(c_t \odot o_t) \tag{6}$$

We can imagine the hidden state as the short-term memory and cell state as the long-term memory of the LSTM network.

The output $\hat{y}_t$ is then computed as:

$$\hat{y}_t = f(W_{h_y} h_t + \vec{b}_y) \tag{7}$$

To see the detailed illustration of LSTM cell see Figure 2.

Presented LSTM architecture in Figure 2 closely maps the state-of-the-art design from (Zaremba et al., 2014). Note that we dropped the network's parameters, matrices of weights, and vectors of biases to keep it well-arranged.

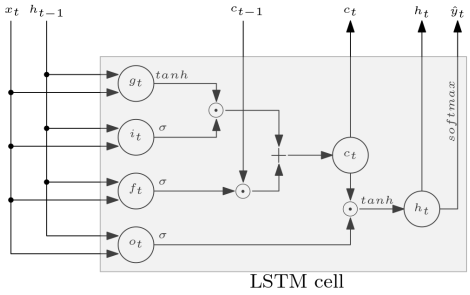


Figure 2: Example of the LSTM architecture.

3.2.1 Bidirectional Long Short-term Memory

The traditional LSTM networks, the same as standard recurrent neural networks and bidirectional recurrent neural networks (BRNNs), are not suitable for some tasks as the hidden and cell states are determined only by prior states. Such tasks include text and speech recognition and many more where the output at a time t depends on the past as well as future inputs or labeling problems where the output is only expected after finishing the whole input sequence (Graves, 2012). **Bidirectional Long Short-Term Memory** or shortly (**BLSTM**) networks try to solve this problem by having connections both from the past and future cells. Input to the BLSTM network is then presented in two rounds, once forwards as with the LSTM network and then in a reversed direction from the back. This architecture was introduced in (Graves and Schmidhuber, 2005).

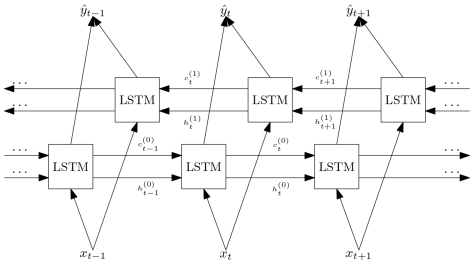


Figure 3: Structure of BLSTM network.

In Figure 3 we can see the structure of BLSTM network, the (0) and (1) in superscripts stand for forwards and backwards directions, respectively. We omitted the detailed representation of LSTM cells to make the illustration simpler.

BLSTM networks can be used to solve similar problems as bidirectional RNNs where we have entire input available beforehand. Training the network in

forward and backward directions helps to gain context from the past and future as well (Brownlee, 2019). In addition, having hidden and cell state enables better storage of information across the timeline even from the distant past or future.

BLSTM networks were found to outperform standard BRNNs in many tasks, e.g. speech recognition. This was proven in the first application of BLSTM networks by Graves et al. for phoneme classification problem (Graves and Schmidhuber, 2005). BLSTMs are not suitable for all tasks, such as where we do not know the final length of the input, and the results are required after each timestamp (*online* tasks).

3.3 Autoencoder

Autoencoder is a type of neural network that can learn a representation of given data by compressing and decompressing the input values. As described in (Chollet, 2016), it consists of two parts, the *encoder* and *decoder*. The encoder is typically a dense feed-forward neural network (other types of neural networks can be used as well) with subsequent layers shrinking in width. The decoder mirrors the structure of the encoder with expanding layers. The autoencoder is then trained with a set of data where input matches the target output. After training, the decoder is detached and the encoder is used as the sole model for prediction. The illustration of the autoencoder setup can be seen in Figure 4.

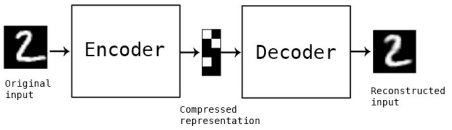


Figure 4: Example of autoencoder for digit compression (Chollet, 2016).

Although the data compressed by autoencoder could be used in image compression, generally autoencoders do not outperform well-known compression algorithms. Since the compression inside autoencoders is not lossless (the output is fuzzy), they are not suitable for practical use of image compression.

Among places where autoencoders found utilization belong dimension reduction and data denoising problems. In dimension reduction, autoencoders are used either as a preprocessing stage in machine learning problems or before data visualization where large data dimension hinders the comprehension of the image. In data denoising, the autoencoder is trained with noisy images as the input and clear pictures being the output. The use of autoencoders is not limited to images, however, they can also be used with audio and other problems affected by noisiness.

4 FEATURE TRANSFORMATIONS USING LSTM NETWORKS

In this section, we present our approach - feature transformation using LSTM networks. We describe feature extraction and preparation, then feature selection along with the central part of this paper, the feature transformation using LSTM networks. Our complete workflow is illustrated in Figure 8 at the end of this section.

4.1 Feature Extraction

For extracting features from PE files, we used Python module `pefile` (Carrera, 2017). This module extracts all PE file attributes into an object from which they can be easily accessed. The structure of the PE files is briefly explained in Section 3.1. We used as many PE attributes as possible and reached the total number of 303 features. Features can be divided into multiple categories based on their origin from the PE file. A summary of all target static features used in our experiments is as follows:

Headers: Data from DOS, NT, File, and Optional headers.

Data Directories: Names and sizes of all data directories. Also adding detailed information from prevalent directories for instance IMPORT, EXPORT, RESOURCE, and DEBUG directories.

Sections: Names, sizes, entropies of all PE sections expressed by their average, min, max, mean and standard deviation. To cooperate with a variable amount of sections in different files, we decided to describe only the first four and last sections individually.

Others: Extra characteristics associated with a file, e.g. byte histogram, printable strings, or version information.

4.2 Feature Preparation

Since not all machine learning models used in our experiments can handle strings and other categorical data, we must such data types encode into numeric values. This strategy is necessary for more than 60 out of 303 columns. We chose to perform common

transformation techniques on the entire dataset as opposed to only using the training set. We believe that by doing so, we can better focus on designing LSTM architectures and our results won't be affected by the capability of other algorithms.

4.2.1 Vectorization

Upfront, we transformed string features into sparse matrix representation using `TfidfVectorizer` from the `scikit-learn` Python library (Pedregosa et al., 2011). This class demands *corpus* (collection of documents) as an input. We also adjusted parameters `stop_words` and `max_df` that influence which words to exclude from further calculations. Among the excluded words are either commonly used words in a given language, words that do not bear any meaning, and words that occur with such high frequencies that they are not statistically interesting for us. To eliminate the massive rise of dimensionality, we set `max_features` parameter according to the feature's cardinality. The transformation itself consists of converting sentences to vectors of token counts. Then they are transformed into tf-idf representation. **Tf-idf** is an abbreviation for the *term frequency times inverse document frequency*. It is a way to express the weight of a single word in the corpus (Maklin, 2019).
Term Frequency is the frequency of a word inside the document. The formula is:

$$\mathrm{tf}(w,d) = \frac{n_{w,d}}{\sum_k n_{k,d}}, \tag{8}$$

where $n_{w,d}$ is the number of times word w appears in a document d and the denominator is the sum of all words found in d.
Inverse Document Frequency is a scale of how much a word is rare across the whole corpus:

$$\mathrm{idf}(w,D) = \log \frac{|D|}{|d \in D : w \in d|} \tag{9}$$

It is a fraction of the total number of documents in corpus D divided by the number of documents containing the specific word.

Tf-idf is then calculated as a multiplication of these two values as follows:

$$\text{tf-idf}(w,d) = \mathrm{tf}(w,d) \cdot \mathrm{idf}(w,D) \tag{10}$$

All of this is done by the aforementioned class `TfidfVectorizer`, and as a result, we get a matrix of tf-idf features that can be used in further computations.

4.2.2 Hashing

For non-string values, we used a technique called **feature hashing**. This approach turns the column of values into a sparse matrix using the value's hash as an index to the matrix. For this task, we used `FeatureHasher` also from the `scikit-learn`. The class takes an optional argument `n_features` which limits the number of columns in the output matrix. We set this argument dynamically according to the size of the feature's value set.

4.3 Feature Selection

Even though we tried to limit the rise of new features, we ended up with 1488 features. To speed up the forthcoming training process, we tried several feature selection techniques to reduce the dimensionality of the dataset.

Before all else, we filled missing values by column's mean and divided data into train and test splits to ensure correct evaluation of the model's performance. For this, we used `train_test_split` from `sklearn.model_selection` with test split taking 20% of the dataset. Afterwards, we transformed features to stretch across a smaller range. For this task, we looked for another class from `sklearn.preprocessing` library and selected `MinMaxScaler`. This scaler turns each feature x to lie between zero and one. The transformation is calculated as:

$$\frac{x_i - \min(x)}{\max(x) - \min(x)} \tag{11}$$

For feature selection, we settled with PCA (Principal Component Analysis) with the number of components determined by testing conducted with the state-of-the-art ML algorithms: AdaBoost, Decision tree, Feed-forward neural network, Random forest, K-nearest neighbours, Support vector machine, Gaussian naive bayes, and Logistic regression. The same ML algorithms were used to evaluate performance of LSTM-based transformation (see Section 5.2). We tested a number of components ranging from 2 to 1400, however increasing benefits were found only until 50 components, after which we did not measure any significant improvements. The results are presented in Figure 5.

Note that while the resulting components are not primarily in the form of sequences, they can still be sequentially processed using the LSTM and achieve solid classification results (see Section 5.3).

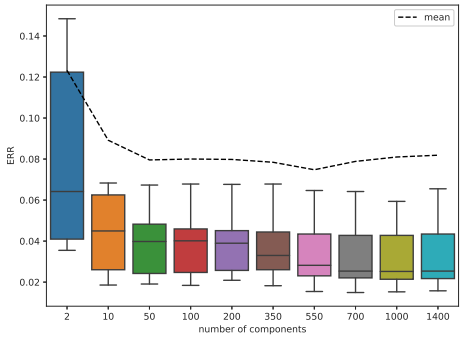


Figure 5: Average error rate (ERR) of ML algorithms across number of components.

4.4 Feature Transformation using LSTM Network

We experimented with various LSTM architectures which we used for feature transformation. All networks were trained only on the train set. After the training process, the train and test set were transformed using the LSTM network.

Our research is not limited to only LSTM networks, however, bidirectional version of LSTM networks (BLSTM) was also included in our experiments. We considered two different types of neural networks: the *Basic version* consisting of one (B)LSTM layer and the *Deep version* with four (B)LSTM layers, each layer containing 50 LSTM units equal to the number of input features. All networks were trained up to 50 epochs with a batch size of 32, Adam optimization, and mean squared error loss function.

4.4.1 Type 1

The first type of LSTM network we experimented is based on autoencoder's architecture. In this case, we worked only with explanatory variables with a network designed to predict the same values which were given on input. The predicted transformation was taken from the penultimate layer's last hidden state. Schema of the Type 1 transformer is illustrated in Figure 6.

4.4.2 Type 2

The second type was similar to the regular use of the LSTM network, where we work with both the explanatory and response variables. For prediction, we used the last hidden state of the penultimate LSTM layer as with Type 1. The last layer was occupied by a single neuron with a sigmoid activation function. Diagram of the Type 2 transformer is presented in Figure 7.

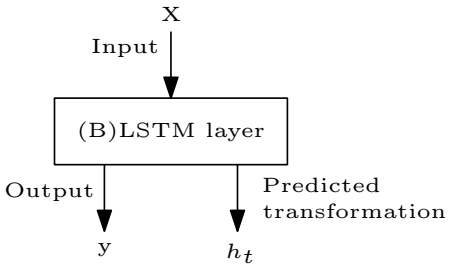


Figure 6: Schema of Basic version Type 1 transformer.

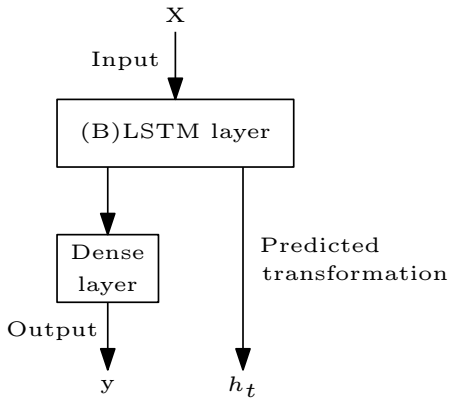


Figure 7: Schema of Basic version Type 2 transformer.

4.4.3 Description of All Transformed Datasets

The following is a description of the datasets used in testing. Recall that "Basic" and "Deep" in the description below denote one layer and four layers of the deep network, respectively:

BLSTM_AE_basic Transformed by Basic version of BLSTM Type 1 (autoencoder) network.

BLSTM_AE_deep Transformed by Deep version of BLSTM Type 1 (autoencoder) network.

BLSTM_basic Transformed by Basic version of BLSTM Type 2 network.

BLSTM_deep Transformed by Deep version of BLSTM Type 2 network.

LSTM_AE_basic Transformed by Basic version of LSTM Type 1 (autoencoder) network.

LSTM_AE_deep Transformed by Deep version of LSTM Type 1 (autoencoder) network.

LSTM_basic Transformed by Basic version of LSTM Type 2 network.

LSTM_deep Transformed by Deep version of LSTM Type 2 network.

VANILLA Control dataset, no transformations made.

4.5 Evaluation using Supervised ML Algorithms

The final part of the experiment workflow consists of evaluation of the aforementioned transformations. We tested several supervised ML algorithms and compared their performance on vanilla and transformed datasets. Detailed description of this part can be found in Section 5.2. Figure 8 overviews the experiment pipeline.

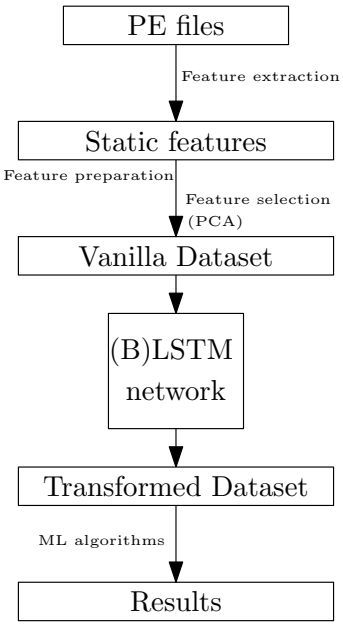


Figure 8: Experiment pipeline.

5 EXPERIMENTS

In this section, we firstly describe the dataset used in our experiments. Then we specify our experimental setup in detail and evaluation methods, and in the end, we present our results.

5.1 Dataset

We gathered a dataset of 30,154 samples which are evenly distributed between malware and benign files. For amassing benign files, we searched disks on university computers and the malware files were obtained from an online repository https://virusshare.com which we thanks for the access.

5.2 Experimental Setup and Evaluation Methods

The performance of LSTM pre-treatment was evaluated by the following supervised ML algorithms. Among the ML algorithms we used were Support vector classification (SVC) with kernel *rbf*, deep Feed-forward network (FNN) with 8 hidden layers (128-128-64-64-32-32-16-16 neurons per layer) all with *ReLU* activation function, trained up to 200 epochs with Adam optimization and binary cross-entropy loss function. Further, we tested Decision tree, Random forest, AdaBoost, K-nearest neighbours (k=5), Gaussian naive bayes, and Logistic regression. The hyperparameters which we did not mention were left to default settings as set by authors of the `scikit-learn` library (Pedregosa et al., 2011) except for the FNN which was modeled with the help of the Python deep learning library `Keras` (Chollet et al., 2015).

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.

5.2.1 Metrics

In this section, we present the metrics we used to measure the performance of our proposed classification models. 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 classifiers on the test set is measured using the following standard metrics:

Accuracy (*ACC*) Proportion of correctly classified samples out of all predictions:

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{12}$$

Error rate (*ERR*) The inverse of accuracy:

$$ERR = 1 - ACC \tag{13}$$

Sensitivity (*TPR, Recall*) How many samples from the positive class were predicted correctly:

$$TPR = \frac{TP}{TP+FN} \tag{14}$$

Fall-out (*FPR*) Probability of predicting samples from the negative class as positives:

$$FPR = \frac{FP}{FP+TN} \tag{15}$$

5.3 Results

In order to expose any biases in the data, we tested the ML algorithms with 5-fold cross-validation using `cross_validate` from `scikit-learn` library.

We found that the results did not only vary between different network architectures but also among particular ML algorithms. These observations are presented in the heatmap in Figure 9. These results indicate that Type 1 based on autoencoder design does not seem to improve the performance whatsoever. However, Type 2, especially deep versions of LSTM and BLSTM networks, seem to enhance the performance of many algorithms significantly.

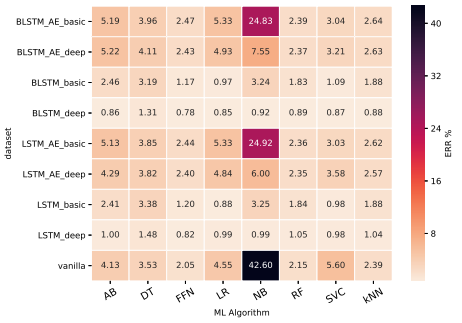


Figure 9: Heatmap comparing the ERR of ML algorithms with respect to different transformer architectures.

Tables 1, 2 and 3 present the improvements made by pre-treatment with LSTM and BLSTM networks for different ML algorithms used for evaluation. Note that the performance of Logistic regression, Naive Bayes, SVC, or AdaBoost algorithms increased the most significantly.

Table 1: Baseline results of ML algorithms on unedited (vanilla) dataset.

ML Algorithm	ACC	TPR	FPR	ROC-AUC
AdaBoost	95.87 ± 0.39	95.49 ± 0.55	3.75 ± 0.46	95.87 ± 0.39
DecisionTree	96.47 ± 0.20	96.37 ± 0.25	3.44 ± 0.40	96.47 ± 0.20
Feed-ForwardNetwork	97.95 ± 0.12	97.75 ± 0.36	1.86 ± 0.23	97.95 ± 0.12
LogisticRegression	95.45 ± 0.38	94.62 ± 0.38	3.73 ± 0.45	95.45 ± 0.38
NaiveBayesGaussian	57.40 ± 16.01	16.63 ± 36.13	1.84 ± 4.11	57.40 ± 16.01
RandomForest	97.85 ± 0.20	97.52 ± 0.24	1.82 ± 0.19	97.85 ± 0.20
SVC(kernel=rbf)	94.40 ± 1.74	93.49 ± 1.74	4.69 ± 1.81	94.40 ± 1.74
kNN(k=5)	97.61 ± 0.21	97.17 ± 0.28	1.94 ± 0.16	97.61 ± 0.21

Table 2: Results of ML algorithms on dataset transformed by deep LSTM network.

ML Algorithm	ACC	TPR	FPR	ROC-AUC
AdaBoost	99.00 ± 0.71	98.76 ± 0.78	0.76 ± 0.66	99.00 ± 0.71
DecisionTree	98.52 ± 0.55	98.43 ± 0.60	1.40 ± 0.50	98.52 ± 0.55
Feed-ForwardNetwork	99.18 ± 0.77	99.10 ± 0.64	0.75 ± 0.90	99.18 ± 0.77
LogisticRegression	99.01 ± 0.72	98.89 ± 0.78	0.87 ± 0.67	99.01 ± 0.72
NaiveBayesGaussian	99.01 ± 0.67	98.70 ± 0.58	0.69 ± 0.78	99.01 ± 0.67
RandomForest	98.95 ± 0.77	98.85 ± 0.77	0.95 ± 0.78	98.95 ± 0.77
SVC(kernel=rbf)	99.02 ± 0.72	98.73 ± 0.74	0.69 ± 0.70	99.02 ± 0.72
kNN(k=5)	98.96 ± 0.75	98.82 ± 0.80	0.91 ± 0.71	98.96 ± 0.75

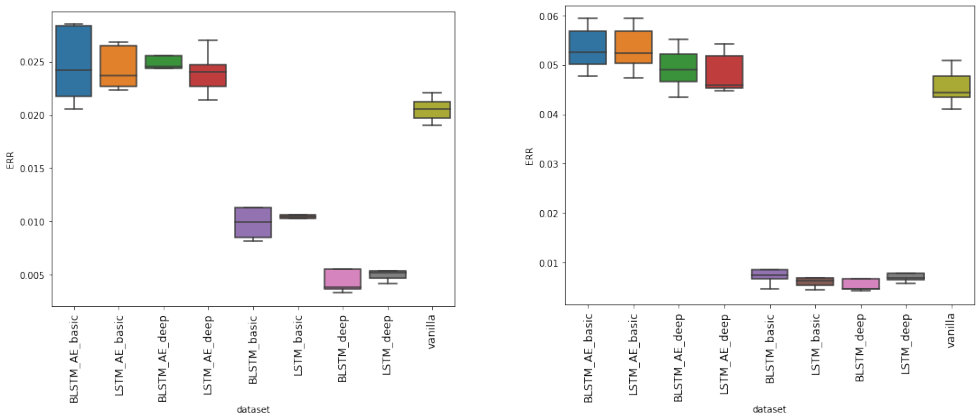
The performance of the two most successful classifiers evaluated on all transformed datasets considered in our experiments, Feed-forward neural network, and Logistic regression, is presented in Figure 10.

To emphasize our results, we express the performance of the ML algorithms in terms of error rate (in [%]). In Table 4, we overview the error rates (ERR) of ML algorithms evaluated on the original and transformed dataset by deep BLSTM network.

6 CONCLUSIONS

We collected a large number of PE binaries from available resources. From these binaries, we extracted as many features as possible, which we later scale down by the feature selection algorithm PCA in order to reduce the dimension of our dataset. After that, we conducted extensive testing with various (B)LSTM network architectures used to transform the selected features. On these transformed datasets, we ran a cross-validation benchmark using multiple supervised ML algorithms to see whether the feature transformation based on (B)LSTM networks can increase the performance of the ML algorithms in comparison to the performance to the accuracy on the vanilla dataset.

We have found that the feature transformation by (B)LSTM nets was hugely successful, decreasing error rate from 58.6% to 97.84% depending on the ML algorithm used. These gains were achieved by so-called Type 2 architecture which was similar to the standard use of recurrent neural networks for classification problems. In contrast, the Type 1 design based on autoencoder structure didn't prove to en-



(a) Feed-forward neural network.

(b) Logistic regression.

Figure 10: Boxplots showing the performance of the two most successfull classifiers evaluated on multiple datasets.

Table 3: Results of ML algorithms on dataset transformed by deep BLSTM network.

ML Algorithm	ACC	TPR	FPR	ROC-AUC
AdaBoost	99.14 ± 0.77	99.06 ± 0.81	0.78 ± 0.80	99.14 ± 0.77
DecisionTree	98.69 ± 0.71	98.67 ± 0.79	1.30 ± 0.65	98.69 ± 0.71
Feed-ForwardNetwork	99.22 ± 0.84	99.12 ± 0.87	0.67 ± 0.83	99.22 ± 0.84
LogisticRegression	99.15 ± 0.78	99.07 ± 0.81	0.78 ± 0.79	99.15 ± 0.78
NaiveBayesGaussian	99.08 ± 0.78	98.65 ± 0.82	0.48 ± 0.77	99.08 ± 0.78
RandomForest	99.11 ± 0.74	99.01 ± 0.80	0.78 ± 0.72	99.11 ± 0.74
SVC(kernel=rbf)	99.13 ± 0.82	98.94 ± 0.87	0.67 ± 0.79	99.13 ± 0.82
kNN(k=5)	99.12 ± 0.82	99.02 ± 0.87	0.78 ± 0.79	99.12 ± 0.82

Table 4: Comparison of the results achieved from the ML algorithms evaluated on the vanilla (non-transformed) dataset and the transformed dataset by deep BLSTM network.

ML Algorithm	ERR (no LSTM)	ERR (with LSTM)	ERR decreased by
AdaBoost	4.13	0.86	79.18
DecisionTree	3.53	1.31	62.89
Feed-ForwardNetwork	2.05	0.78	61.95
LogisticRegression	4.55	0.85	81.32
NaiveBayesGaussian	42.60	0.92	97.84
RandomForest	2.15	0.89	58.60
SVC(kernel=rbf)	5.60	0.87	84.46
kNN(k=5)	2.39	0.88	63.18

hance performance. The transformation by Type 2 deep transformers brought all tested ML algorithms to a similar level. The smaller performance increments were observed among the ML algorithms which already performed well on the non-transformed dataset.

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

AV-test (2019). Security report 2018/19. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf.

Barron, D. W. (1978). *Assemblers and loaders*. Elsevier Science Inc.

Brownlee, J. (2019). How to develop a bidirectional lstm for sequence classification in python with keras. https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/.

Carrera, E. (2017). Pefile. https://github.com/erocarrera/pefile.

Chollet, F. (2016). Building autoencoders in keras. https://blog.keras.io/building-autoencoders-in-keras.html.

Chollet, F. et al. (2015). Keras. https://keras.io.

Graves, A. (2012). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 13–39. Springer.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610.

Hardy, W., Chen, L., Hou, S., Ye, Y., and Li, X. (2016). Dl4md: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining (DMIN)*, page 61. The Steering Committee of The World Congress in Computer Science, Computer ....

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Karl Bridge, M. (2019). Pe format - win32 apps. "https://docs.microsoft.com/en-us/windows/win32/debug/pe-format".

Kowalczyk, K. (2018). Portable executable file format. https://blog.kowalczyk.info/articles/pefileformat.html.

Latto, N. (2020). What is wannacry? https://www.avast.com/c-wannacry.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2020). *Mining of massive data sets*. Cambridge university press.

Lipton, Z. C., Berkowitz, J., and Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, pages 5–25.

Lu, R. (2019). Malware detection with lstm using opcode language. *arXiv preprint arXiv:1906.04593*.

Maklin, C. (2019). Tf idf: Tfidf python example. https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Saxe, J. and Berlin, K. (2015). Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE.

Symantec, C. (2019). Internet security threat report 2019. https://docs.broadcom.com/doc/istr-24-2019-en.

Vinayakumar, R., Soman, K., Poornachandran, P., and Sachin Kumar, S. (2018). Detecting android malware using long short-term memory (lstm). *Journal of Intelligent & Fuzzy Systems*, 34(3):1277–1288.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, pages 1–3.

Zhou, H. (2018). Malware detection with neural network using combined features. In *China Cyber Security Annual Conference*, pages 96–106. Springer.

4.4 Paper 4 - Improving Classification of Malware Families using Learning a Distance Metric

Mgr. Martin Jureček (70%), Mgr. Olha Jurečková (20%), prof. Ing. Róbert Lórencz, CSc. (10%)
In *Proceedings of 7th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 643-652, Virtual event, 2021

The main topic of this paper is a multiclass classification problem where each malware family and benign files have their own class. Three distance metric learning algorithms were employed to learn the Mahalanobis distance metric to improve the multiclass classification performance of the KNN classifier. Paper provides practical information concerning performance, computational time, and resource usage.

The practical use of distinguishing between malware families lies in helping malware analysts deal with a large number of samples.

The paper was written in cooperation with Olha Jurečková, who worked on experiments concerning the low-dimensional representation of malware and benign files.

Improving Classification of Malware Families using Learning a Distance Metric

Martin Jureček, Olha Jurečková and Róbert Lórencz
Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
{*jurecmar, jurecolh, lorencz*}*@fit.cvut.cz*

Keywords: Malware Family, PE File Format, Distance Metric Learning, Machine Learning.

Abstract: The objective of malware family classification is to assign a tested sample to the correct malware family. This paper concerns the application of selected state-of-the-art distance metric learning techniques to malware families classification. The goal of distance metric learning algorithms is to find the most appropriate distance metric parameters concerning some optimization criteria. The distance metric learning algorithms considered in our research learn from metadata, mostly contained in the headers of executable files in the PE file format. Several experiments have been conducted on the dataset with 14,000 samples consisting of six prevalent malware families and benign files. The experimental results showed that the average precision and recall of the k -Nearest Neighbors algorithm using the distance learned on training data were improved significantly comparing when the non-learned distance was used. The k -Nearest Neighbors classifier using the Mahalanobis distance metric learned by the Metric Learning for Kernel Regression method achieved average precision and recall, both of 97.04% compared to Random Forest with a 96.44% of average precision and 96.41% of average recall, which achieved the best classification results among the state-of-the-art ML algorithms considered in our experiments.

1 INTRODUCTION

A large number of new malicious samples are created every day, which makes manual analysis impractical. The majority of these samples are generated by malware generators, which need to input some parameters. These malware generators, together with their particular settings, define corresponding malware families. Samples generated from the same generator with a fixed setting (i.e., from the malware family) may be potentially similar to each other and different from samples belonging to other malware families or benign files. This work focuses on leveraging these differences to distinguish between malware families. Note that samples from the same malware family, however, generated in a different time period may be different from each other (Wadkar et al., 2020).

Since the samples are usually obfuscated, it is difficult to classify new (previously unseen) samples into the correct malware families. Moreover, there is no known general similarity measure suitable for a feature set extracted from the PE file format to correctly cluster all malware families. (Jureček and Lórencz, 2018) presented distance metric specially designed for the PE file format that can handle all data types of features.

Our work focuses on the multiclass classification problem where each malware family and benign files have their own class. This multiclass classification problem is more challenging than a binary classification problem where the goal is to distinguish between malicious and benign files. However, the results of (Basole et al., 2020) may indicate that an increasing number of families (from 2 to 20 families) drops an average balanced accuracy slightly.

The practical use of distinguishing between malware families lies in helping malware analysts to deal with a large number of samples. Due to a large number of malicious files that come to antivirus vendors, there is a need to automatically categorize malware into groups corresponding to malware families. Samples belonging to the same group are similar to each other with respect to some similarity measure (determined by distance metric). These groups are then distributed to malware analysts and assuming that files belonging to the same group have similar behavior, it may help speed up the further analysis.

Usually, malware analysts are specialists for some limited number of malware families. If we assume

that samples were classified correctly and samples of the same family are similar to each other and dissimilar to the samples of other families, using our approach, the analysts can focus only on those samples which belong to the malware families for which the analysts are specialized.

Good similarity measure plays an important role in the performance of distance-based classifiers, such as k -Nearest Neighbors (KNN). 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. This is the goal of distance metric learning methods used to learn the parameters of distance metrics from training data. As a result, they can potentially improve the performance of the classifiers.

In our experiments, we consider six malware families, which is a relatively small number. Another limitation of our work lies in assuming that our dataset is large enough for training distance metric learning (DML) algorithms. However, in practice, new families or new malware variants are continuously emerging. Therefore the training set, at some moment, may not contain enough samples of the desired malware family to train some supervised learning classifier.

The contributions of this paper are as follows:

- We determined and described the list of 25 features all extracted (except one, i.e., size of a file) from the PE file format. For each feature from a section header, we considered the order of the section rather than the type of the section (such as .text, .data, .rsrc, etc.). While the sections' order turns out to be important for malware detection, this kind of information is often not mentioned in research papers.
- Using three DML algorithms, LMNN, NCA, and MLKR, we achieved significantly better multiclass classification results than any state-of-the-art ML algorithms considered in our experiments. We provided practical information concerning performance, computational time, and resource usage.
- We showed that the DML-based methods might improve multiclass classification results even when standard methods such as feature selection or algorithm tuning were already applied. As a result, we suggest using DML algorithms as an important preprocessing step.

The rest of the paper is organized as follows. In Section 2, we review recent malware detection methods based on machine learning focusing on the classification of malware families. In Section 3, we give some theoretical background and discuss three distance metric learning techniques used in our experiments. The experimental setup and results of feature selection algorithms are presented in Section 4. Section 5 describes DML-based experiments and results. We summarize our research work in Section 6.

2 RELATED WORK

This section briefly reviews the previous research papers on malware family classification related to our work.

In (Basole et al., 2020), the authors conducted experiments based on byte n-gram features, and they considered 20 malware families. A binary classification were performed on different levels. In the first level, for each of 20 families, they performed binary classification for 1,000 malware samples from one family and 1,000 benign samples. In the second level, the malware class consists of two malware families; in the third level, the malware class consists of three malware families, and so on up to level 20, where the malware class contains all of the 20 malware families. The authors applied four state-of-the-art machine learning algorithms: KNN, Support Vector Machines, Random Forest, and Multilayer Perceptron. The best classification results (balanced accuracy) was achieved using KNN and Random Forest, over 90% (at level 20), while KNN achieves the most consistent results.

A fully automated system for analysis, classification, and clustering of malware samples was introduced in (Mohaisen et al., 2015). This system is called AMAL and it collects behavior-based artifacts describing files, registry, and network communication, to create features that are then used for classification and clustering of malware samples into families. The authors achieved more than 99% of precision and recall in classification and more than 98% of precision and recall for unsupervised clustering.

In (Ahmadi et al., 2016), the authors proposed a malware classification system using different malware characteristics to assign malware samples to the most appropriate malware family. The system allows the classification of obfuscated and packed malware without doing any deobfuscation and unpacking processes. High classification accuracy of 99.77% was achieved on the publicly accessible Microsoft Malware Challenge dataset.

(Islam et al., 2013) presented a classification method based on static (function length frequency and printable sting) and dynamic (API function names with API parameters) features that were integrated

into one feature vector. The obtained results showed that integrating features improved classification accuracy significantly. The highest weighted average accuracy was achieved by the meta-Random Forest classifier.

Another malware family classification system referred to as VILO is presented in (Lakhotia et al., 2013). They used TFIDF-weighted opcode mnemonic permutation features and achieved between 0.14% and 5.42% fewer misclassifications using KNN classifier than does the usage of n-gram features.

In the rest of this section, we survey some previous works on distance metric learning applied to the problem of malware detection. There is only a couple of works that address this topic. (Jureček and Lórencz, 2020) deals with measure learning and its application to malware detection. Particle swarm optimization (PSO) was used to find appropriate feature weights for the heterogeneous distance function used in the KNN classifier. Positions of particles in the initialization step of PSO were set according to the information gain computed in the feature selection step rather than randomly. As a result, PSO was accelerated, and better classification accuracy was achieved using the weighted distance function.

Work (Kong and Yan, 2013) concerns with a malware detection method based on structural information. The discriminant distance metric is learned to cluster the malware samples belonging to the same malware family.

3 BACKGROUND

Performance of some ML classifiers, such as KNN, depends significantly on the distance metric used to compute similarity measure between two samples. These classifiers rely on the assumption that samples belonging to the same class are close to each other (with respect to the distance function), and they are far from samples belonging to the different classes.

The DML algorithms were designed to improve the performance of distance-based classifiers via learning the distance metric. This section provides background and a brief description of three state-of-the-art distance metric learning algorithms, LMNN, NCA, and MLKR, used in our experiments.

Euclidean distance is by far the most commonly used distance metric. Let $\mathbf{x}$ and $\mathbf{y}$ be two n-dimensional feature vectors. The weighted Euclidean distance is defined as follows:

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

where w_i is a weight (non-negative real number) associated with the jth feature. The distance metric learning problem for weighted Euclidean distance is defined as finding (or learning) an appropriate weight vector $w = (w_1, \ldots, w_n)$ using training data, with respect to some optimization criterion, usually minimizing error rate.

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

Mahalanobis distance for two n-dimensional feature vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

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

where $\mathbf{M}$ is a positive semidefinite matrix. Mahalanobis distance can be considered as a generalization of Euclidean distance, since if $\mathbf{M}$ is the identity matrix, then $d_{\mathbf{M}}$ in Eq. (2) is reduced to Euclidean distance. If $\mathbf{M}$ is diagonal, this corresponds to learning the feature weights $M_{ii} = w_i$ from Eq. (1) defined for weighted Euclidean distance.

The goal of learning the Mahalanobis distance is to find an appropriate matrix $\mathbf{M}$ with respect to some optimization criterion. In the context of the KNN classifier, the goal is to find a matrix $\mathbf{M}$, which is estimated from the training set, which leads to the lowest error rate of the KNN classifier. Since a positive semidefinite matrix $\mathbf{M}$ can always be decomposed as $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, distance metric learning problem can be viewed as finding either $\mathbf{M}$ or $\mathbf{L} = \mathbf{M}^{\frac{1}{2}}$. Mahalanobis distance defined in Eq. (2) expressed in terms of the matrix $\mathbf{L}$ is defined as

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

The matrix $\mathbf{L}$ can be used to projects the original feature space into a new embedding feature space. This projection is a linear transformation defined for feature vector $\mathbf{x}$ as

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

Note that the Mahanalobis distance $d_{\mathbf{L}}(\mathbf{x},\mathbf{y})$ for two samples from the original feature space equals the Euclidean distance $d(\mathbf{x}',\mathbf{y}') = \sqrt{(\mathbf{x}'-\mathbf{y}')^\top (\mathbf{x}'-\mathbf{y}')}$

in the space transformed by Eq. (4). This transformation is usefull since computation of Euclidean distance has lower time complexity than computation of Mahalanobis distance.

In the rest of this paper, we will consider the feature space as a real n-dimensional space $\mathbb{R}^n$. The following subsections briefly describe three distance metric learning methods that we used in our experiments.

3.1 Large Margin Nearest Neighbor

Large Margin Nearest Neighbor (LMNN) (Weinberger et al., 2006) is one of the state-of-the-art distance metric learning algorithms used to learn a Mahalanobis distance metric for KNN classification. LMNN consists of two steps. In the first step, for each instance, $\mathbf{x}$, a set of k nearest instances belonging to the same class as $\mathbf{x}$ (referred to as *target neighbors*) is identified. In the second step, we adapt the Mahalanobis distance with the goal that the target neighbors are closer to $\mathbf{x}$ than instances from different classes that are separated by a large margin.

The Mahalanobis distance metric is estimated by solving a semidefinite programming problem defined as:

$$\min_{\mathbf{L}} \sum_{i,j:j\rightarrow i} \Big(d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_j)^2 + \\ + \mu \sum_{k:y_i\neq y_k} \max\Big(0, 1 + d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_j)^2 - d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_k)^2\Big)\Big) \tag{5}$$

The notation $j \rightarrow i$ 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.

3.2 Neighborhood Component Analysis

(Goldberger et al., 2005) proposed the Neighborhood Component Analysis (NCA), a distance metric learning algorithm specially designed to improve KNN classification.

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

$$p_{ij} = \frac{\exp(-||\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_j||_2^2)}{\sum_{l\neq i}\exp(-||\mathbf{L}\mathbf{x}_i - \mathbf{L}\mathbf{x}_l||_2^2)}, \qquad p_{ii} = 0 \tag{6}$$

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

$$\arg\max_{\mathbf{L}} \sum_{i=0}^{N-1} \sum_{j:j\neq i, y_j = y_i} p_{ij} \tag{7}$$

The well-known gradient ascent algorithm is used to solve this optimization problem. Note that both LMNN and NCA algorithms do not make any assumptions on the class distributions.

3.3 Metric Learning for Kernel Regression

(Weinberger and Tesauro, 2007) proposed Metric Learning for Kernel Regression (MLKR), which aims at training a Mahalanobis matrix by minimizing the error loss over the training samples:

$$\mathcal{L} = \sum_i (y_i - \hat{y}_i)^2 \tag{8}$$

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

$$\hat{y}_i = \frac{\sum_{j\neq i} y_j K(\mathbf{x}_i,\mathbf{x}_j)}{\sum_{j\neq i} K(\mathbf{x}_i,\mathbf{x}_j)} \tag{9}$$

MLKR can be applied to many types of kernel functions $K(\mathbf{x}_i,\mathbf{x}_j)$ and distance metrics $d(\mathbf{x},\mathbf{y})$.

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

4 EXPERIMENTAL SETUP

In this section, we present our dataset, describe evaluation measures and feature selection results.

4.1 Dataset

Our experiments are based on the dataset containing 14,000 samples consisting of 6 malware families and benign files. The dataset is well-balanced since each of the 6 malware families is of equal size, i.e., 2,000 samples, and the number of benign files is also 2,000. The malicious programs were obtained from (VirusShare, 2020), an online repository containing various malware families. Benign files were gathered from university computers. We confirm that all malicious samples considered in our experiments match

known signatures from antivirus companies. Also, none of our benign programs was detected as malware.

In our experiments, we used the following six prevalent malware families:

Allaple – a polymorphic network worm that spreads to other computers and performs denial-of-service (DoS) attacks.

Skeeyah – a Trojan horse that infiltrates systems and steals various personal information and adds the computer to a botnet.

Virlock – ransomware that locks victims' computer and demands a payment to unlock it.

Virut – a virus with backdoor functionality that operates over an IRC-based communications protocol.

Vundo – a Trojan horse that displays pop-up advertisements and also injects JavaScript into HTML pages.

Zbot – also known as Zeus, is a Trojan horse that steals configuration files, credentials, and banking details.

4.2 Evaluation Measures

In this section, we present the metrics we used to measure the performance of the classification models. In a binary classification problem, 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 binary classifiers considered in our experiments is measured using three standard metrics. The most intuitive and commonly used evaluation metric is the error rate:

$$\text{ERR} = \frac{\text{FP}+\text{FN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{10}$$

It is defined on a given test set as the percentage of incorrectly classified samples. Alternative for error rate is accuracy defined as ACC$=1-$ERR. The second metric is precision, and it is defined as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{11}$$

Precision is the percentage of samples classified as malware that are truly malware. The third parameter, recall (or true positive rate), is defined as:

$$\text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{12}$$

Recall is the percentage of truly malicious samples that were classified as malware.

In the multiclass evaluation, since all the classes have the same number of samples, we use averaged versions of error rate, precision and recall. Average error rate is defined as follows:

$$\text{(average) ERR} = \frac{1}{N}\sum_{i\leq N} \mathbf{1}_{\text{class}_{\text{pred}} \neq \text{class}_{\text{true}}} \tag{13}$$

where N is the size of our dataset, and $\mathbf{1}$ is the indicator function. Average precision and average recall is defined as an average resulting precisions and recalls, respectively, across all classes.

4.3 Feature Selection

The features used in our experiments are extracted from the portable executable (PE) file format (Microsoft, 2019), which is the file format for executables, DLLs, object code, and others used in 32-bit and 64-bit versions of the Windows operating system. The PE file format is the most widely used file format for malware samples run on desktop platforms.

For extracting features from PE files, we used Python module `pefile` (Carrera, 2017). This module extracts all PE file attributes into an object from which they can be easily accessed. We extracted 358 numeric features that are based on static information only, i.e., without running the program. The dimensionality is high since for each section, and for each kind of characteristics (array of flags), we consider each flag as a single feature.

Before applying feature selection methods, all features were normalized using procedure `preprocessing.normalize` from the Scikit-learn library (Scikit-learn, 2020). We then employed the six feature selection methods also imported from the Scikit-learn library.

Table 1 shows error rates of the KNN ($k=1$) classifier applied to the feature set reduced using the corresponding feature selection algorithms. The lowest error rate of 4.13% was achieved for 25 selected features by RFE Logistic Regression. The KNN for the original feature set (i.e., 358 features) achieved an error rate of 4.31%. All feature selection algorithms were evaluated by 5-fold cross-validation on the randomly chosen training data containing 80% of the whole dataset. The remaining 20% of the dataset was

Table 1: Evaluation of the feature selection algorithms in terms of error rates of the KNN ($k = 1$) classifier. The abbreviation SFM refers to Scikit-learn procedure `feature_selection.SelectFromModel`. The abbreviation RFE refers to Recursive Feature Elimination implemented in `feature_selection.RFE` from the Scikit-learn library as well.

	Error rates [%] for specific number of features					
Feature selection method	2	5	10	25	50	75
Principal component analysis	11.00	5.30	4.85	4.22	4.26	4.31
SFM Logistic Regression	13.61	7.90	4.76	4.20	4.29	4.31
SFM Decision Tree	26.56	8.44	6.21	4.72	4.50	4.37
Information Gain	15.17	8.77	7.79	5.61	4.31	4.52
RFE Logistic Regression	17.08	6.49	5.30	4.13	4.29	4.31
RFE Decision Tree	11.67	7.53	4.76	4.63	4.65	4.22

reserved for testing the DML and ML algorithms (see Section 5).

The following Fig. 1 illustrates the performance of RFE Logistic regression for a various number of features. For 50 and more features, the corresponding error rates are approximately the same as the error rate achieved from the original feature set (i.e., 358 features).

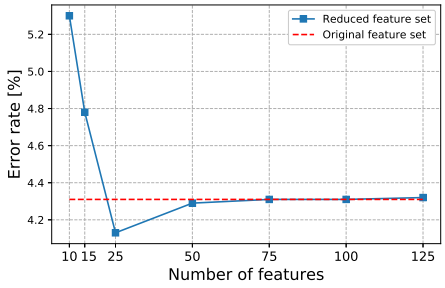


Figure 1: Relation between dimensionality and error rate of KNN classifier ($k = 1$).

Notice that for each feature selection algorithm under consideration, except SFM Decision Tree, we achieved a surprisingly low error rate with only two features, as shown in Table 1. We will provide more experiments for this extremely low dimensionality in Section 5.2.

We also performed all six feature selection algorithms to explore an even higher number of features: 100, 125, and 150. However, the corresponding error rates achieved from all the feature selection algorithms were higher than 4.13%. As a result, in all experiments (except those in Section 5.2), we will consider 25 features selected by RFE Logistic regression algorithm. To make our results reproducible, Table 2 summarizes all features used in our experiments. We keep the name of the fields in the same form as in the documentation (Microsoft, 2015), in order that the reader can easily find the detailed description.

The field file_size (size of the file on disk) is not contained within the PE structure, and note that it differs from the field SizeOfImage, which is the size of the image loaded in memory.

PE files are divided into one or more sections. The sections contain code, data, imports, and various characteristics. PE section features are considered separately for each section. The order of the sections is not the same for each PE file. Moreover, malware authors can change the order of the sections. Therefore, we prefer to consider only the order of sections (rather than their names). The special importance among all sections of a PE file has the last one since it may contain useful information, especially for some types of malware, such as 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.

5 EXPERIMENTAL RESULTS

This section presents multiclass classification results based on six base ML classifiers: k-Nearest Neighbor ($k = 1$), Logistic Regression, (Gaussian) Naive Bayes, Random Forest (number of trees in the forest = 100), and Multilayer Perceptron (hidden layer sizes=(200,100), maximum number of iterations = 300, activation function = 'relu', solver for weight optimization = 'adam', random number generation for weights and bias initialization = 1). Implementations of the DML algorithms, the ML classifiers, and the classification metrics, are based on the Scikit-learn library (Scikit-learn, 2020). If not mentioned, the hyperparameters of the ML classifiers and the DML methods were set to their default values as set in the Scikit-learn library.

The input feature vectors used in the following experiments are described in Section 4.3, and its dimensionality is 25, except in the experiment described in Section 5.2 where we consider only two-dimensional

Table 2: List of 25 features selected by the RFE Logistic Regression algorithm sorted by importance. All except one (file size) are extracted from the PE file format.

Position	Feature name	Structure
1.	PointerToRawData	Section Header (the first section)
2.	Characteristics, flag IMAGE_SCN_TYPE_DSECT	Section Header (the first section)
3.	Characteristics, flag IMAGE_SCN_TYPE_COPY	Section Header (the first section)
4.	Characteristics, flag IMAGE_SCN_TYPE_NOLOAD	Section Header (the first section)
5.	RVA of Exception Table	Optional Header Data Directories
6.	PointerToRawData	Section Header (the third section)
7.	Characteristics, flag IMAGE_SCN_TYPE_GROUP	Section Header (the first section)
8.	RVA of Certificate Table	Optional Header Data Directories
9.	RVA of Base RelocationTable	Optional Header Data Directories
10.	RVA of Bound Import	Optional Header Data Directories
11.	VirtualSize	Section Header (the first section)
12.	SizeOfRawData	Section Header (the first section)
13.	Size of file	not part of PE format
14.	VirtualAddress	Section Header (the fourth section)
15.	VirtualSize	Section Header (the second section)
16.	RVA of Import Table	Optional Header Data Directories
17.	Characteristics, flag IMAGE_SCN_TYPE_REG	Section Header (the first section)
18.	AddressOfEntryPoint	Optional Header Standard Fields
19.	RVA of Export Table	Optional Header Data Directories
20.	VirtualAddress	Section Header (the first section)
21.	RVA of TLS Table	Optional Header Data Directories
22.	PointerToRawData	Section Header (the second section)
23.	VirtualAddress	Section Header (the last section)
24.	Size of Import Table	Optional Header Data Directories
25.	VirtualAddress	Section Header (the second section)

feature vector.

All the following experiments were 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.

5.1 Application of Distance Metric Learning Algorithms

In the first set of experiments, we applied the KNN classifier using the following four distances: Euclidean (non-learned) distance, and three Mahalanobis distances learned by three DML algorithms (separately) LMNN, NCA, and MLKR. We used 5-fold cross-validation as follows. The training dataset (80 % of the whole dataset) was randomly divided into five subsets of equal size, where four subsets were used for training the DML algorithms, and one subset was used for testing. First, the DML algorithm was trained on the four subsets, and then KNN with learned Mahalanobis distance metric was employed using training and testing sets. This procedure was run five times with different subset reserved for testing. Classification results obtained for each fold are then averaged to produce a single cross-validation estimate.

The first experiment focuses on the hyperparameter k that expresses the number of nearest neighbors considered in the KNN classifier and LMNN (the learning rate of the optimization procedure = 10^{-6}) algorithm. Note that both MLKR nor NCA do not depend on k. Fig. 2 shows the effect of k on the error rate of the KNN using all four distances.

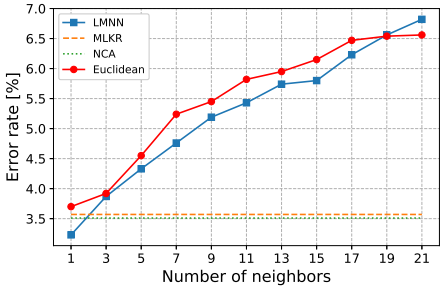


Figure 2: The relation between the number of nearest neighbors (k) and error rate (ERR) compared for four distances.

The KNN classifier for $k = 1$ with Euclidean distance achieved the ERR of 3.70%. Error rates of the KNN using Mahalanobis distance learned by DML algorithms are equal to: 3.23% for LMNN, 3.51% for NCA, and 3.57% for MLKR.

The result of distance metric learning algorithms is $n \times n$ matrix, where n is the dimension of the feature vector. Since the number of components of the matrix to be learned grows at a quadratic rate 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. In the next experiment, we used the Principal component analysis to reduce the data's dimension and examine the learning ability of distance metric learning algorithms. We applied 5-fold cross-validation technique described above. For our fixed-size training dataset, the highest performance improvement gained by using DML algorithms was achieved for the following dimensions: $n = 3$ for LMNN, $n = 4$ for NCA, and $n = 6$ for MLKR. These results may indicate that considering 25-dimensional feature vectors used in our experiments, our dataset's size is insufficient for learning as many as 625 parameters (i.e., the number of components of the matrix $\mathbf{M}$).

We also explored the variation of classification results based on DML algorithms across six malware families and benign class. Precisions and recalls of the KNN classifier using the hyperparameter $k = 1$ are summarize in Table 3. The results indicate that the precisions and recalls corresponding to malware families depend significantly on the DML algorithms.

Regarding resource usage of DML algorithms, 32GB of RAM was sufficiently enough (i.e., without a need to use the disk as swap memory) for all conducted experiments. The average computational times of the DML algorithms applied on 8,960 samples are as follows: LMNN took 550 seconds, NCA took 960 seconds, and MLKR took 4,800 seconds.

5.2 Representation of Malware Families in Two Dimensions

In this section, we examine the classification results based on two-dimensional feature vectors. Two of the most important features, PointerToRawData and flag IMAGE_SCN_TYPE_DSECT (see Table 2), were selected by the RFE Logistic Regression and used for multiclass classification.

Fig. 3 presents the effectiveness of the KNN ($k = 1$) classification of malware families and benign files represented by only two-dimensional feature vectors. While we achieved 99% accuracy for the malware families Skeeyah and Virlock, benign files' accuracy was 58%.

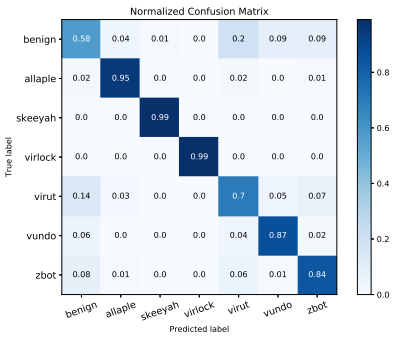


Figure 3: Normalized confusion matrix comparing the accuracy of KNN ($k = 1$) using Euclidean distance for six malware families and benign samples.

Two-dimensional representation of feature vectors allows us to show malware families as points in the plane. Three malware families, Allaple, Virut, and Vundo, are illustrated in Fig. 4. One hundred samples were chosen randomly from each of these classes, and we achieved the average classification accuracy of 92.00%.

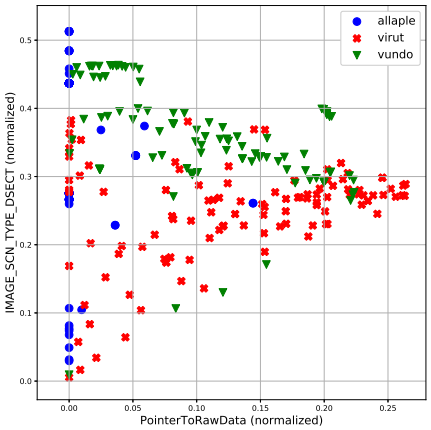


Figure 4: Three malware families: Allaple, Virut, and Vundo represented in two dimensions.

5.3 Comparison with the State-of-the-Art ML Algorithms

In the last experiment, LMNN, NCA, and MLKR methods (each separately) were used to transform the data, as it is shown in Eq. (4) in Section 3. We compared the performance of the state-of-the-art ML algorithms for original (non-transformed) data and for the transformed data. Table 4 shows that the high-

Table 3: KNN ($k = 1$) classification results of particular malware families and class of benign samples.

	Precision [%]				Recall [%]			
class	Euclid	LMNN	NCA	MLKR	Euclid	LMNN	NCA	MLKR
allaple	98.58	98.79	92.32	92.39	98.42	98.49	92.60	91.69
benign	94.48	94.24	99.24	99.09	92.08	92.15	97.47	97.47
skeeyah	99.36	98.45	98.88	98.41	98.89	98.30	99.04	99.04
virlock	98.79	99.55	99.72	100	99.39	99.40	99.15	99.15
virut	97.24	96.51	97.38	96.19	95.44	97.07	96.49	96.49
vundo	95.38	97.26	94.85	95.27	98.07	97.71	98.43	98.11
zbot	92.65	92.59	93.03	93.50	94.21	94.34	92.33	93.08
Averaged	96.64	96.77	96.49	96.41	96.64	96.78	96.50	96.43

Table 4: Classification results of DML algorithms evaluated for several state-of-the-art ML algorithms.

	Average precision [%]						Average recall [%]					
	KNN	LR	NB	DT	RF	MLP	KNN	LR	NB	DT	RF	MLP
original	96.15	86.38	82.98	94.76	96.44	96.22	96.14	86.17	77.79	94.78	96.41	96.17
LMNN	96.77	89.55	82.02	95.20	97.05	96.39	96.78	89.27	81.03	95.20	97.00	96.35
NCA	96.45	90.78	78.08	94.87	96.76	95.11	96.46	90.60	75.02	94.86	96.72	95.07
MLKR	97.04	87.94	77.96	95.15	97.05	96.50	97.04	88.12	75.41	95.13	97.02	96.49

est average precision of 97.05% was achieved using Random Forest on the data transformed by LMNN or by MLKR. The KNN classifier achieved the highest average recall of 97.04% on the data transformed by MLKR.

Regarding original (non-transformed) data, the highest average precision of 96.44% and the highest average recall of 96.41% among base ML algorithms were both achieved by Random Forest.

Focusing on the KNN ($k = 1$) classifier, any of the three DML methods achieved better classification results (both average precision and recall) than the KNN classifier using common (non-learned) Euclidean distance. Regarding the Naive Bayes classifier, we achieved a very low precision of 54.53% for Vundo and a very low recall of 47.02% for Zbot compared to other ML classifiers. These two results cause that average precision and recall for Naive Bayes are significantly lower compared to other ML algorithms.

6 CONCLUSIONS

In this paper, we employed three distance metric learning algorithms to learn the Mahalanobis distance metric to improve multiclass classification performance for our dataset containing six prevalent malware families and benign files. We classified the previously unseen samples using the KNN classifier with the learned distance and achieved significantly better results than using common (non-learned) Euclidean distance. The classification results demonstrate that DML-based methods outperform any of the state-of-the-art ML algorithms considered in our experiments. Our results indicate that the classification performance based on DML methods could be further improved if we use a larger dataset for training the distance metric. Another experiment was concerned with low-dimensional representations of the input feature space. We achieved surprisingly good classification results even for two-dimensional feature vectors.

In future work, other types of features, such as byte sequences, opcodes, API and system calls, and others, could be used and possibly improve the classification results. It would also be interesting to explore the application of distance metric learning algorithms to clustering into 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

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., and Giacinto, G. (2016). Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, pages 183–194.

Basole, S., Di Troia, F., and Stamp, M. (2020). Multifamily malware models. *Journal of Computer Virology and Hacking Techniques*, pages 1–14.

Carrera, E. (2017). Pefile. https://github.com/erocarrera/pefile.

Goldberger, J., Hinton, G. E., Roweis, S. T., and Salakhutdinov, R. R. (2005). Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520.

Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, 36(2):646–656.

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

Jureček, M. and Lórencz, R. (2020). Distance metric learning using particle swarm optimization to improve static malware detection. In *Int. Conf. on Information Systems Security and Privacy (ICISSP)*, pages 725–732.

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.

Lakhotia, A., Walenstein, A., Miles, C., and Singh, A. (2013). Vilo: a rapid learning nearest-neighbor classifier for malware triage. *Journal of Computer Virology and Hacking Techniques*, 9(3):109–123.

Microsoft (2019). Pe format - win32 apps. https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

Microsoft (Revision 9.3, 2015). Visual studio, microsoft portable executable and common object file format specification.

Mohaisen, A., Alrawi, O., and Mohaisen, M. (2015). Amal: High-fidelity, behavior-based automated malware analysis and classification. *computers & security*, 52:251–266.

Scikit-learn (2020). scikit-learn.org. https://scikit-learn.org/.

VirusShare (2020). Virusshare.com. http://virusshare.com/.

Wadkar, M., Di Troia, F., and Stamp, M. (2020). Detecting malware evolution using support vector machines. *Expert Systems with Applications*, 143:113022.

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.

Weinberger, K. Q. and Tesauro, G. (2007). Metric learning for kernel regression. In *Artificial Intelligence and Statistics*, pages 612–619.

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.

4.5 Paper 5 - Application of Distance Metric Learning to Automated Malware Detection

Mgr. Martin Jureček (80%), prof. Ing. Róbert Lórencz, CSc. (20%)
This paper was submitted to the journal *IEEE Access* on 2nd February.

This paper presents the malware detection model based on distance metric learning. The detection system processed the data from PE file format where numeric features are normalized and nominal features are turned to conditional probabilities. Training samples were firstly used to train distance metric, and then the same samples were used in KNN classifier with already learned distance metric to classify testing samples.

The paper describes the proposed approach based on the KNN classifier using weighted Euclidean distance learned by the modification of the PSO algorithm that was proposed in Paper 2. This approach was compared with three distance metric learning algorithms applied in experiments from Paper 4.

In addition, the paper focuses on the problem of detecting as much malware as possible while keeping a low false positive rate. Insufficiently low false positive error is considered seriously in the antivirus industry. An optimization criterion based on an error rate, and modification of the LMNN method, are proposed to penalize false positives.

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Application of Distance Metric Learning to Automated Malware Detection

MARTIN JUREČEK[1] and RÓBERT LÓRENCZ[1]
[1]Department of Information Security, Faculty of Information Technology, Czech Technical University in Prague, 160 00 Prague, Czech Republic
Corresponding author: Martin Jureček (e-mail: jurecmar@fit.cvut.cz).

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".

ABSTRACT The goal of distance metric learning is 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 applied distance metric learning for the problem of malware detection. We focused 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 proposed a malware detection system using Particle Swarm Optimization that finds the feature weights to optimize the similarity measure. We compared the performance of the approach with three state-of-the-art distance metric learning techniques. We found 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 PE file format. Our experimental results showed that our malware detection system based on distance metric learning achieves a 1.09% of error rate at 0.74% of false positive rate (FPR) and outperformed all machine learning algorithms considered in the experiment. Considering the second task related to keeping minimal FPR, we achieved a 1.15% of error rate at only 0.13% of 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 detriment 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 are 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, on the other hand 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 detriment, such as damaging a computer system and not 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. 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. Finding the most appropriate parameters of the metric concerning some optimization criterion is typically formulated as an optimization problem. Evolutionary algorithms, swarm

IEEE Access

algorithms, and other heuristics [3] are suitable for 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 concerning 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 [4]. In this work, we empirically demonstrated how to apply distance metric learning on 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 in the Windows environment. Features consist of metadata from portable executable (PE) file format [5]. Our proposed detection model is based only on static malware analysis, aiming to search for information about the file structure without running a program. While the static analysis can be evaded by anti-malware 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.

High malware detection accuracy is not the only goal of the classification models from the antivirus vendors' perspective. In practice, a false positive error is considered seriously since benign software owners could complain that their product was detected as malware, which potentially can ruin their business. For this reason, we proposed an optimization criterion that takes into account the cost of false positives.

The main contributions of our work are as follows:

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

Scalable optimization criterion for PSO-based model: To consider the higher cost of false positive, we constructed a cost function called *weighted error rate* which we used 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 compared these models with the PSO-based model and provided practical information concerning performance, computational time, and resource usage. We showed that the DML-based methods might improve malware classification results even when standard methods such as feature selection or algorithm tuning were already 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 we 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

This section briefly reviews some recent works related to malware detection based on machine learning techniques. We mainly focus on works using static analysis of Windows PE files.

The application of machine learning techniques to malware detection has been an active research area for about the last twenty years. Researchers have tried to apply various well-known techniques such as Neural Networks, Decision Trees, Support Vector Machines, ensemble methods, and many other popular machine learning algorithms. Recent survey papers [6], [7] provide comprehensive information on malware detection techniques using machine learning algorithms.

Wadkar *et al.* [8] proposed the system based on static features extracted from PE files for detecting evolutionary modifications within malware families. Support Vector Machines (SVM) models were trained over a sliding time window, and the differences in SVM weights were quantified using χ^2 statistic. For most of all 13 malware families considered in the experiments, the system detected significant changes.

Yang and Liu [9] proposed a detection model called TuningMalconv with two layers: raw bytes model in the first layer and 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.* [10]. The detection model is based on semi-supervised transfer learning and was deployed on the cloud as a SaaS (Software as a Service). The detection model was evaluated on Kaggle malware datasets, and it improved classification accuracy from 94.72% to 96.90%.

Xue *et al.* [11] proposed a classification system Malscore, which combine 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 k-Nearest Neighbors. The authors performed experiments on more than 170,000

malware samples from 63 malware families and achieved 98.82% of malware classification accuracy.

Zhong and Gu [12] improved the 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 a specific malware family. As a result, the MLDLS 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 work proposed by Raff *et al.* [13] was from the PE header, more specifically, from MS-DOS, COFF, and Optional Header. Neural networks were learned from raw bytes, which were not parsed to explicit features, and as a result, no preprocessing nor feature engineering was required. More than 400,000 samples were used for training, and the Fully Connected Neural Network model achieved the highest accuracy.

Kolosnjaji *et al.* [14] proposed neural network architecture that combines convolutional and feedforward neural layers. The authors used only static malware analysis where inputs to feedforward layers were fields of the PE header, while inputs to convolutional layers were assembly opcode sequences. The proposed hybrid neural network achieved 93% on precision and recall.

Surprisingly, there is a lack of experimentation with distance metric learning techniques applied on large and real-world data sets 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 the two following works. In [15], we applied the Particle Swarm Optimization algorithm to the problem of finding the appropriate feature weights used in the heterogeneous distance function [16] specially 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 [4] focused on the application of three distance metric learning methods applied to multiclass classification problem with seven classes: six prevalent malware families and benign files. Using Metric Learning for Kernel Regression method to learn Mahalanobis distance metric, we achieved average precision and recall, both of 97.04%, evaluated with k-Nearest Neighbors 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 follows:

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

The goal of learning the weighted Euclidean distance is to find an appropriate weight vector $w = (w_1, \ldots, w_n)$ concerning some optimization criterion, usually minimizing error rate. Several other distance functions have been presented [17]. In order to improve results, many weighting schemes were proposed. A review of feature weighting methods for lazy learning algorithms was proposed in [18].

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

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

where $\mathbf{M}$ is a positive semidefinite matrix. Mahalanobis distance can be considered as a generalization of Euclidean distance, since if $\mathbf{M}$ is the identity matrix, then $d_{\mathbf{M}}$ in Eq. (2) is reduced to common Euclidean distance. If $\mathbf{M}$ 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 an appropriate matrix $\mathbf{M}$ concerning some optimization criterion. Regarding the KNN classifier, the goal can be defined as to find a matrix $\mathbf{M}$ which is estimated from the training set, leading to the lowest error rate of the KNN classifier. Since a positive semidefinite matrix $\mathbf{M}$ can always be decomposed as $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, distance metric learning problem can be viewed as finding either $\mathbf{M}$ or $\mathbf{L} = \mathbf{M}^{\frac{1}{2}}$. Mahalanobis distance defined in Eq. (2) expressed in terms of the matrix $\mathbf{L}$ is defined as

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

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 $\mathbf{x}, \mathbf{y}$ from the original space defined in Eq. (2) corresponds to the Euclidean distance between transformed points $\mathbf{x}' = \mathbf{L}\mathbf{x}, \mathbf{y}' = \mathbf{L}\mathbf{y}$ defined as follows:

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

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

Distance metric learning has attracted a lot of attention in the machine learning field and is still an active research area [19]. There have been proposed many methods [20], [21], [22]. Next, we briefly describe three state-of-the-art distance metric learning methods that we used in our experiments. In this work, weighted Euclidean distance was learned by

the Particle Swarm Optimization algorithm, and 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) [23] is one of the state-of-the-art distance metric learning algorithms used to learn a Mahalanobis distance metric for KNN classification. LMNN consists of two steps. In the first step, for each instance, $\mathbf{x}$, a set of k nearest instances belonging to the same class as $\mathbf{x}$ (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 $\mathbf{x}$ than instances from different classes separated by a large margin. The Mahalanobis distance metric is estimated by solving semidefinite programming problem defined as:

$$\min_{\mathbf{L}} \sum_{i,j:j\rightsquigarrow i} \Big(d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_j)^2 + \\ + \mu \sum_{k:y_i\neq y_k} [1 + d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_j)^2 - d_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_k)^2]_+ \Big) \quad (6)$$

The notation $j \rightsquigarrow i$ reffers 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 distances between samples $\mathbf{x_i}$ and their target neghbors $\mathbf{x_j}$,
2) to maximize distances between samples $\mathbf{x_i}$ and their impostors $\mathbf{x_k}$ which are samples which belong among the nearest neighbors of $\mathbf{x_i}$ and 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 [24], 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.* [25] proposed the Neighborhood Component Analysis (NCA), which is a distance metric learning algorithm specially designed to improve KNN classification.

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

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

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

$$\arg\max_{\mathbf{L}} \sum_i \sum_{j:j\neq i, y_j = y_i} p_{ij} \quad (8)$$

The gradient ascent algorithm solves this optimization problem. Both LMNN and NCA algorithms do not make any assumptions on the class distributions.

C. METRIC LEARNING FOR KERNEL REGRESSION

Weinberger *et al.* proposed Metric Learning for Kernel Regression (MLKR) [26], which aims at training a Mahalanobis matrix by minimizing the error loss over the training samples:

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

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

$$\hat{y}_i = \frac{\sum_{j\neq i} y_j K(\mathbf{x}_i,\mathbf{x}_j)}{\sum_{j\neq i} K(\mathbf{x}_i,\mathbf{x}_j)} \quad (10)$$

MLKR can be applied apply to many types of kernel functions $K(\mathbf{x}_i,\mathbf{x}_j)$ and distance metrics $d(\mathbf{x}_i,\mathbf{x}_j)$.

Recall that the mentioned distance metric learning algorithms can be used as supervised dimensionality reduction algorithms. Considering the matrix $\mathbf{L} \in \mathbb{R}^{d\times n}$ with $d < n$ then the dimension of transformed sample $\mathbf{x}' = \mathbf{L}\mathbf{x}$ 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 portable executable (PE) file format [5], which is the file format for executables, DLLs, object code, and others used in 32-bit and 64-bit versions of the Windows operating system. PE file format is the most widely used file format for malware samples 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 followed immediately by the section headers, which provide information about sections, including 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 order 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 special importance among all sections of a PE file has the last one. It may contain useful information, especially for some types of malware, such as 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 $T = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_m, c_m)\}$ be the training set, where $\mathbf{x}_i$ is a feature vector and $c_i = cl(\mathbf{x}_i)$ 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 is 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 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 minimal, resp. 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 applied several feature selection and extraction algorithms and selected the most relevant features.

B. FINDING THE FEATURE WEIGHTS USING PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) [27] 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.

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

- 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.
- 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 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: fitness function f, a training set T_{pso}, and vector p of feature importance scores [28] achieved from 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. Operation $\otimes$ denotes 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 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 global search while lower values tend to local search. Parameters ϕ_1 and ϕ_2 represent the weights, and they 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 clas-

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

sifier. 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 stuck into the local minima.

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

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

where $|T_{test}|$ is 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 false positive:

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

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

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

One point of view on the WERR criterion is that we agree to "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 Fig. 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 testing set. The training set is used for training a distance metric and classifier.

Step 2: **Parsing binaries.** For each sample, we extract and store information from PE file format. These features will be preprocessed in the following two steps, and relevant features only 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 appeared 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.** Feature selection algorithm is used to determine the relevant features and produced the final version of the feature set.

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

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

Steps 7: **Evaluation:** Performance metrics are used to measure classification results.

Computing the conditional probabilities for nominal features and performing feature selection algorithm for all three types of features are done only to the training samples. The corresponding conditional probabilities and selected features are applied to design 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.

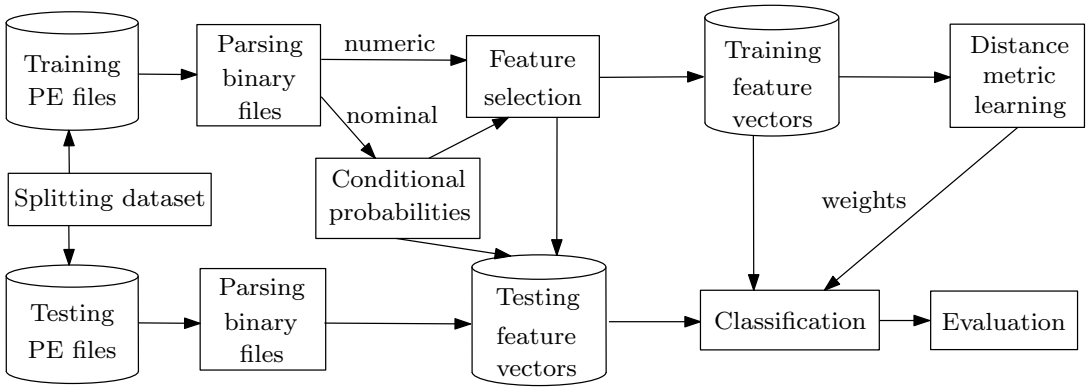


FIGURE 1: Architecture of our proposed malware detection model using distance metric learning.

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 are malicious, and 75,167 are benign programs. The malicious and benign programs were obtained from the industrial partner's laboratory, and the Virusshare repository [29]. We confirm that all malicious samples considered in our experiments match known signatures from anti-virus companies. Also, none of our benign programs was detected as malware.

For extracting features from PE files, we used Python module `pefile` [30]. This module extracts all PE file attributes into an object from which they can be easily accessed. We extracted 370 features based on static information only, i.e., without running the program. The dimensionality is high since, in each section, flags of each kind of characteristic (i.e., an array of flags) were considered as single features.

Our implementations of the feature selection algorithms, DML algorithms, the ML classifiers, and the classification metrics are based on the Scikit-learn library [31]. If not mentioned, the hyperparameters of the ML classifiers and the DML methods were set to their default values as set in the Scikit-learn library.

Our implementation was executed on a single computer platform having 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. Note that memory usage was not exceeded in each 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. Alternative for ERR is accuracy defined as ACC $=$ 1$-$ERR. The second parameter, True Positive Rate (TPR) (or detection rate), is defined as:

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

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 wrongly 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. Fig. 2 shows that the highest accuracy was achieved by the Recursive Feature Elimination (RFE) Logistic Regression for 75 selected features. 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 6

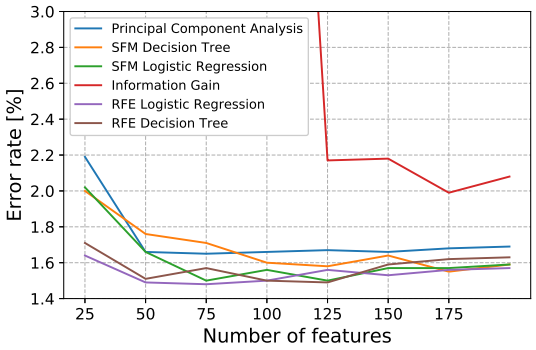


FIGURE 2: Evaluation of the feature selection algorithms in terms of error rates of the KNN ($k = 3$) classifier. The abbreviation SFM refers to procedure `feature_selection.SelectFromModel`, and the abbreviation RFE refers to Recursive Feature Elimination implemented in `feature_selection.RFE`, both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

in Appendix A summarizes 75 selected features used in our experiments. We kept the name of the fields in the same form as in the documentation [32] so that the reader can easily find the 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 false positive rate, and finally, we compared our approach based on PSO with 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 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 next experiments. We left the hyper-parameters of NCA and MLKR in the default values as stated 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 are as follows: 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 PSO parameters were chosen following the guidelines from [33].

We run the PSO algorithm ten times, and Fig. 3 illustrates the mean and standard deviation of error rate corresponding to the various number of iterations. The PSO algorithm was run with 50 iterations, and however, for 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.

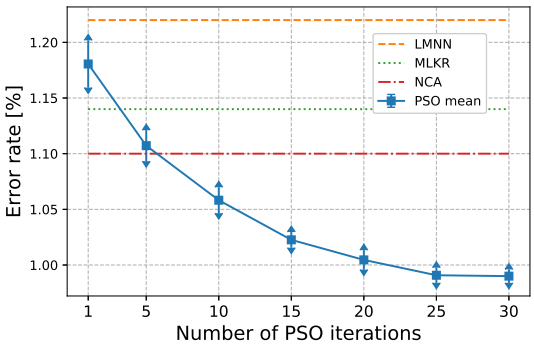


FIGURE 3: Average classification error with the standard deviation is illustrated as a function of the maximum number of iteration in the PSO algorithm.

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 1 shows the performance of the KNN classifier ($k = 3$) using common Euclidean distance, Mahalanobis distance learned by three selected DML algorithms, and 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

TABLE 1: 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**

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 between 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 dataset keeping the ratio of their sizes 70:30. The number of training samples, learning method, and the average learning times, TPR, FPR, and ERR estimated on the testing set are summarized in Table 2. The number of nearest neighbors considered in both LMNN-based and PSO-based models was $k = 3$.

For smaller datasets (i.e., 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) Effect of parameter k

We discuss how different parameter settings of k (i.e., number of neighbors) affects the performance of the KNN classifier. We explored the variation of error rates for the following three variants: Euclidean distance (i.e., 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 Fig. 4 shows the averaged results of the KNN classifier for various values of the parameter k from the set $\{1, 3, 5, \ldots, 21\}$.

In the additional experiment, we explored the relation between the number of neighbors and learning time. Fig. 5

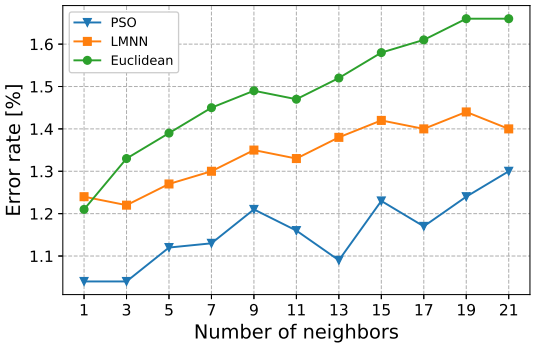


FIGURE 4: The relation between the number of neighbors and the performance of KNN using various distances.

shows that with the increasing number of neighbors learning time of PSO increases only negligibly compared to the learning time of LMNN.

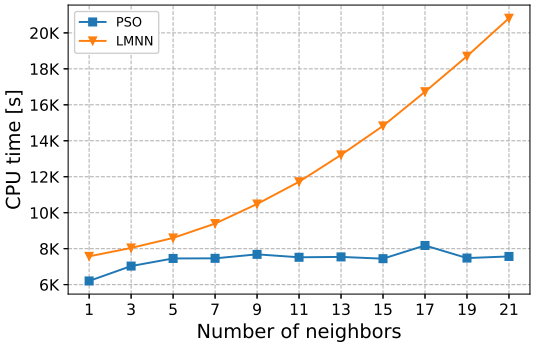


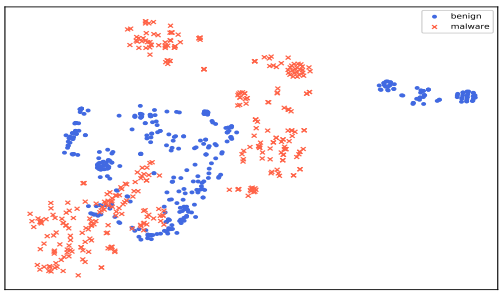
FIGURE 5: Learning time of LMNN and PSO (with 30 iterations). The experiment was conducted ten times, and the computational times were averaged.

3) LMNN projection of original feature space

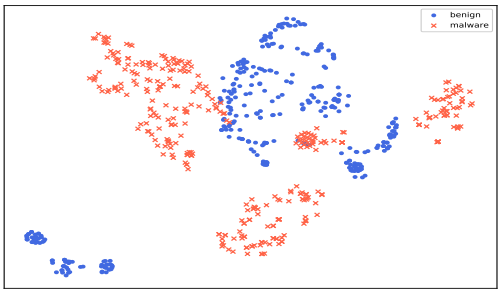
In the next experiment, we used the weight matrix $\mathbf{L}$, defined in Eq. (3), learned by LMNN to project the original feature space into a new embedding feature space following the goal that the k-Nearest Neighbors of each instance belong to the same class while a large margin separates instances from different classes. Recall that this projection is linear transformation defined as $\mathbf{x}' = \mathbf{L}\mathbf{x}$. This experiment aims to illustrate the difference between original (non-transformed) data and transformed data using LMNN. Two-dimensional embedding of 700 samples using t-SNE algorithm [34] is shown in Fig. 6 where similarity plots for four scenarios are compared.

TABLE 2: Comparison of the LMNN-based model and PSO-based model for various sizes of datasets.

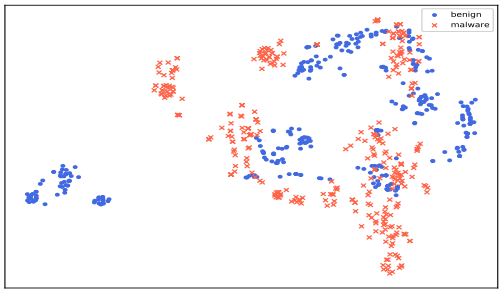
# train samples	Learning method	Learning time	TPR[%]	FPR[%]	ERR[%]
1000	LMNN	8s	96.26	4.17	3.95
	PSO	9s	95.92	1.44	2.79
5000	LMNN	2min 34s	96.51	1.90	2.71
	PSO	2min 32s	97.41	1.97	2.29
20000	LMNN	23min 51s	97.94	1.50	1.78
	PSO	18min 11s	98.39	1.49	1.55
50000	LMNN	2h 6min 40s	97.88	0.32	1.22
	PSO	1h 5min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



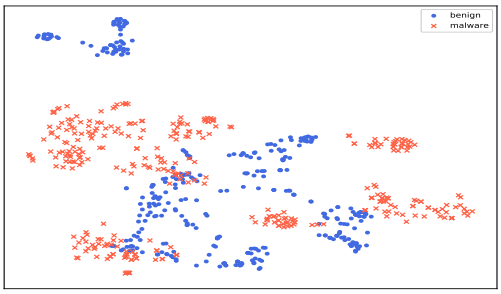
(a) Dataset A (error rate = 3.29%)



(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, and 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 by the KNN classifier ($k = 3$).

4) Limitation of LMNN for larger datasets

The result of the DML algorithms for the Mahalanobis distance metric is $n \times n$ 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 weighted Euclidean distance needs only n parameters to be learned.

In the next experiment, we used the Principal Component Analysis [35] to reduce the data's dimension and examine the learning ability of the LMNN-based model for various dimensions of 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 (non-learned) Euclidean distance. Our fixed-size dataset consisted

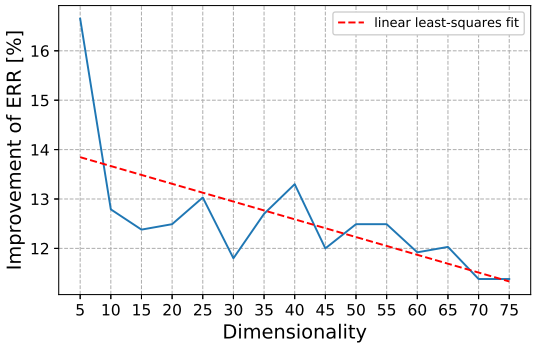


FIGURE 7: The relation between the dimensionality of the feature vector and *performance improvement* achieved using the KNN classifier with Mahalanobis distance metric learned by LMNN.

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

C. MINIMIZING OF 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 error rate. In this experiment, we performed PSO with WERR optimization criterion for $c \in \{1, \ldots, 10\}$. Fig. 8 presents the relation between the coefficient c and false positive rate and error rate achieved by the KNN classifier ($k = 3$). 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 Mahanalobis distance produces only 20 to 30 false positives. WERR criterion in the PSO algorithm does not affect such a low number of false positives. Concerning 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, until $c \leq 8$.

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 benign samples for every 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 static and dynamic analysis.

Regarding 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 denotes the set of k target neighbors of x_i. Then the modification of LMNN concerning minimizing false positives is as follows:

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

where η_k denotes cost of false positive and it is define as follows:

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

Similar 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 3 shows the performance of the modified LMNN according to Eq. (21), and PSO methods with the WERR criterion.

TABLE 3: Results of the experiment with modified LMNN and PSO-based method with emphasis on the 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

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 the *Our proposed method* which refers to the KNN classifier with weighted Euclidean distance where the weights were learned by PSO algorithm with WERR criterion.

A list of machine learning classifiers considered, together with implementation details, is presented in Table 4. We briefly describe the machine learning techniques applied in the experiment.

The k-Nearest Neighbors classifier [36] is one of the most popular supervised learning methods. It is a non-parametric

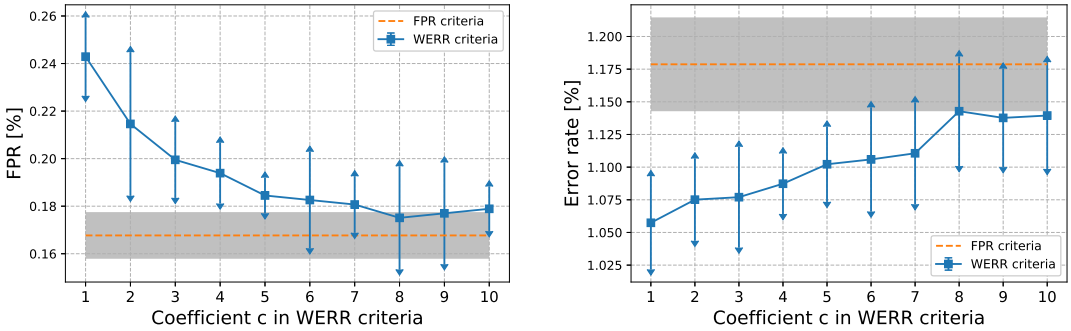


FIGURE 8: The relation between the coefficient c defined in the WERR criterion and false positive rate and error rate.

TABLE 4: 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

method that assigns a class label to each testing sample by a majority vote of its k nearest neighbors.

Support Vector Machine method (SVM) [37] is mainly defined for two-class classification problems. The main idea is to maximize the margin, which is the smallest distance between the training data and the decision boundary. SVM method can also be applied in multiclass classification problems using a binary classifier in a one-against-all situation.

Logistic Regression [38] is a parametric binary classifier that estimates the coefficients from the training data using maximum-likelihood estimation. Similar to SVM, the One-against-all strategy can also be applied to multiclass classification problems.

Naïve Bayes classifier [39] is a probabilistic algorithm based on Bayes' theorem that predicts the class with the highest a posteriori probability. Naïve Bayes classifier is based on the assumption that the features are conditionally independent of one another, which is often not valid in practice.

Decision Tree classifier [40] is represented as a tree, where internal nodes correspond to features and leaf nodes correspond to class labels. Edges leading to children node correspond to feature' values. The feature vector determines the path from the root node to the leaf node.

Deep Neural network [41] is a feedforward 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 [42] is one of the most popular boosting algorithms. It performs 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 [43] is an ensemble learning method that combines the results made by several decision trees using a voting mechanism.

Table 5 provides average classification results of selected supervised machine learning algorithms compared with results of *Our proposed method* defined as the KNN classifier using weighted Euclidean distance learned by the PSO algorithm as described in Section IV.

TABLE 5: 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

All machine learning algorithms were run 20 times for randomly chosen training set 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 significantly higher FPR.

VII. CONCLUSIONS

This paper proposed a malware detection system based on the k-Nearest Neighbor classifier using 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 static analysis for Windows PE files. This approach can also be applied to executable formats of other operating systems, such as MAC OS or Linux.

In addition, we focused on the problem of detecting as much malware as possible while keeping a low false positive rate. Insufficiently low 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 learned feature weights of weighted Euclidean distance, we achieved 0.13% of false positive rate with an error rate of 1.15%.

Ongoing work is focused in the following 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 even lower FPR on an acceptable error rate level.

REFERENCES

[1] D. Salomon, *Foundations of computer security*. Springer Science & Business Media, 2006.

[2] K. Monnappa, *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*. Packt Publishing Ltd, 2018.

[3] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013.

[4] M. Jureček, O. Jurečková, and R. Lórencz, "Improving classification of malware families using learning a distance metric," in *Int. Conf. on Information Systems Security and Privacy (ICISSP)*, 2021, to be published.

[5] Microsoft, "Pe format - win32 apps," Accessed: Jan. 29, 2021. [online], Available: https://docs.microsoft.com/en-us/windows/win32/debug/pe-format.

[6] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.

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

[8] M. Wadkar, F. Di Troia, and M. Stamp, "Detecting malware evolution using support vector machines," *Expert Systems with Applications*, vol. 143, p. 113022, 2020.

[9] L. Yang and J. Liu, "Tuningmalconv: malware detection with not just raw bytes," *IEEE Access*, vol. 8, pp. 140 915–140 922, 2020.

[10] X. Gao, C. Hu, C. Shan, B. Liu, Z. Niu, and H. Xie, "Malware classification for the cloud via semi-supervised transfer learning," *Journal of Information Security and Applications*, vol. 55, p. 102661, 2020.

[11] D. Xue, J. Li, T. Lv, W. Wu, and J. Wang, "Malware classification using probability scoring and machine learning," *IEEE Access*, vol. 7, pp. 91 641–91 656, 2019.

[12] W. Zhong and F. Gu, "A multi-level deep learning system for malware detection," *Expert Systems with Applications*, vol. 133, pp. 151–162, 2019.

[13] E. Raff, J. Sylvester, and C. Nicholas, "Learning the pe header, malware detection with minimal domain knowledge," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 121–132.

[14] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, "Empowering convolutional networks for malware classification and analysis," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 3838–3845.

[15] M. Jureček and R. Lórencz, "Distance metric learning using particle swarm optimization to improve static malware detection," in *Int. Conf. on Information Systems Security and Privacy (ICISSP)*, 2020, pp. 725–732.

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

[17] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of artificial intelligence research*, vol. 6, pp. 1–34, 1997.

[18] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms," *Artificial Intelligence Review*, vol. 11, no. 1-5, pp. 273–314, 1997.

[19] J. L. Suárez, S. García, and F. Herrera, "A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges," *Neurocomputing*, 2020, to be published.

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

[21] B. Kulis *et al.*, "Metric learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2013.

[22] A. Bellet, A. Habrard, and M. Sebban, "A survey on metric learning for feature vectors and structured data," *arXiv preprint arXiv:1306.6709*, 2013.

[23] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in neural information processing systems*, 2006, pp. 1473–1480.

[24] K. Q. Weinberger and L. K. Saul, "Fast solvers and efficient implementations for distance metric learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1160–1167.

[25] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," in *Advances in neural information processing systems*, 2005, pp. 513–520.

[26] K. Q. Weinberger and G. Tesauro, "Metric learning for kernel regression," in *Artificial Intelligence and Statistics*, 2007, pp. 612–619.

[27] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4. Citeseer, 1995, pp. 1942–1948.

[28] M. Kuhn, K. Johnson *et al.*, *Applied predictive modeling*. Springer, 2013, vol. 26.

[29] VirusShare, "Accessed: Jan. 29, 2021. [online]," Available: http://virusshare.com.

[30] E. Carrera, "Pefile," Accessed: Jan. 29, 2021. [online], Available: https://github.com/erocarrera/pefile.

[31] Scikit-learn, "Accessed: Jan. 29, 2021. [online]."

[32] Microsoft, "Microsoft portable executable and common object file format specification," 1999.

[33] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: an overview," *Soft Computing*, vol. 22, no. 2, pp. 387–408, 2018.

[34] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[35] L. I. Smith, "A tutorial on principal components analysis," 2002.

[36] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[37] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[38] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[39] M. E. Maron and J. L. Kuhns, "On relevance, probabilistic indexing and information retrieval," *Journal of the ACM (JACM)*, vol. 7, no. 3, pp. 216–244, 1960.
[40] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.
[41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[42] R. E. Schapire and Y. Freund, "Boosting: Foundations and algorithms," *Kybernetes*, 2013.
[43] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

.

APPENDIX A FEATURES USED IN EXPERIMENTS

Table 6 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 [32], in order that the reader can easily find the detailed description. A detailed description of these features can be found in the documentation [5].

MARTIN JUREČEK graduated from the Charles University in Prague, Faculty of Mathematics and Physics, with a specialization in mathematical methods of information security. He is now a Ph.D. student at the Faculty of Information Technology of the Czech Technical University in Prague. His main research interests focus on the application of machine learning and artificial intelligence approaches to malware detection. Other areas of his interest are algebraic cryptanalysis and the security of cryptocurrencies.

RÓBERT LÓRENCZ graduated from the Faculty of Electrical Engineering of the Czech Technical University in Prague in 1981. He received his Ph.D. degree in 1990 from the Institute of Measurement and Measuring Methods, Slovak Academy of Sciences in Bratislava. Currently he is Full Professor at the Faculty of Information Technology of the Czech Technical University in Prague. His research interests are cryptography and arithmetic units for cryptography primitives, various cryptoanalysis methods of block and stream ciphers. Another topic of his interest is alternative arithmetic for numerical computation.

TABLE 6: 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

Chapter 5

Conclusions

Nowadays, antivirus vendors face several problems concerning malware detection. The concept of employing machine learning to malware detection provides promising solutions. Thousands of papers dealing with machine learning methods for malware detection were proposed. Moreover, since malware developers create more and more sophisticated techniques, it is necessary to use the latest techniques from machine learning to keep error rate and false positive rate as low as possible. This game can converge to the point when artificial intelligence of attackers will fight against the artificial intelligence of malware researchers.

Our work can have practical application since all the malware detection systems mentioned in this thesis are based on static analysis that is significantly faster than dynamic analysis, which involves running the program. Each of our proposed malware detection models can be used as a component of a more complex system relying on results from both static and dynamic analysis.

The proposed static malware detection systems are relatively easy to implement and can be utilized to support commercial antivirus systems. We achieved the most significant results using a forward neural network, which we applied to PE features transformed using a BLSTM network. Specifically, we achieved an accuracy of 99.22% with a 0.67% false positive rate.

5.1 Contributions of the Dissertation Thesis

Contributions of our work are as follows:

Paper 1. We designed the heterogeneous distance metric specially defined for the PE file format. We proposed a classifier combining the weighted k-Nearest Neighbor method and the statistical scoring technique. The results indicate that the proposed heterogeneous distance metric is appropriate for malware detection and that combination of the classifiers may provide a potential benefit to detect samples not detected by WKNN.

Paper 2. We modified the heterogenenous distance function by considering the feature weights. We then applied the PSO algorithm to the problem of finding the most appropriate feature weights. The results indicate that the classification performance of KNN can be improved significantly by using appropriate weights of the features.

Paper 3. Using various LSTM and Bidirectional LSTM network architectures, we transformed the PE features and trained other supervised machine learning algorithms on transformed dataset dataset to improve classification accuracy. Transformation by deep (4 hidden layers) versions of LSTM and BLSTM networks decreased the error rate of several state-of-the-art machine learning algorithms significantly.

Paper 4. We showed that the DML-based methods might improve multiclass classification results even when standard methods such as feature selection or algorithm tuning were already applied. Using three DML algorithms, LMNN, NCA, and MLKR, we achieved significantly better multiclass classification results than any state-of-the-art ML algorithms considered in our experiments.

Paper 5. We proposed the architecture of the malware detection model based on distance metric learning. The detection system processed the PE file format data where numeric features are normalized and nominal features are turned to conditional probabilities. To consider the higher cost of false positive, we constructed a cost function called weighted error rate which we used as a fitness function in the PSO algorithm to minimize error rate and false positive rate.

Finally, we try to evaluate our effort and answer whether we reached the goals presented in Section 1.3. Let recall the first two goals that are related to the binary classification problem. If we accept the error rate that is less than 1% and the false positive rate is also under 1%, then we can say that we reached both goals. In Paper 3, using Feed-Forward Network on dataset transformed by deep BLSTM network, we achieved 99.22% of accuracy at only 0.67% of false positive rate. Note that this result is competitive in comparison to the state-of-the-art. However, to achieve a fair comparison, the same conditions, such as benchmarking dataset, have to be ensured.

Regarding the third goal of the thesis, in Paper 4 and Paper 5, we experimentally verify the usefulness of DML methods. Paper 4 deals with malware family classification. The k-Nearest Neighbors classifier using the Mahalanobis distance metric learned by the Metric Learning for Kernel Regression method achieved average precision and recall, both of 97.04% compared to Random Forest with a 96.44% of average precision and 96.41% of average recall, which achieved the best classification results among the state-of-the-art ML algorithms considered in our experiments. Paper 5 deals with malware detection. Our experimental results showed that our malware detection system based on distance metric learning achieves a 1.09% of error rate at 0.74% of false positive rate (FPR) and outperformed all machine learning algorithms considered in the experiment.

We claim that we fulfilled the fourth goal since in each of our papers, Paper 1 to Paper 5, we applied and described automated processes such as extraction of features, preprocessing, feature selection, training, and evaluating.

5.2 Future Work

The author of the dissertation thesis suggests exploring the following main issues and challenges:

Adversarial learning. Machine learning models are vulnerable to adversarial attacks that can fool the models. For instance, an adversary can craft malware that has a similar feature vector to some benign file's feature vector. As a result, the training set may have different statistical distribution than the distribution of the testing set. The goal is to proposed defense techniques in order for machine learning algorithms can resist such adversarial attacks.

Interpretability of the models. Several malware detection models based on machine learning techniques, such as neural networks, are considered as a black box in the sense that it is difficult (for humans) to determine exactly the reason why a given false positive or false negative occurs. Malware researchers preferred interpretable detection systems, such as rule-based methods, since they can better understand and control. The goal of this challenge is to improve the interpretability of classification models.

Minimizing of reaction time. The majority of new malicious samples are generated by malware generators that need to input some parameters. Since attackers have all antivirus products at their disposal, a common strategy of attackers is to change the setting of malware generators and generate samples as long as none of the antiviruses detect them. New malware is then spread among users by some infection vector. The time period between spreading the malware and creating detection rule, referred to as reaction time, should be minimal since users are not protected during this time period.

Bibliography

[1] Monnappa, K. *Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware*. Packt Publishing Ltd, 2018.

[2] Singh, J.; Singh, J. Challenge of malware analysis: malware obfuscation techniques. *International Journal of Information Security Science*, volume 7, no. 3, 2018: pp. 100–110.

[3] AV-TEST Institute. AV-TEST Security Report 2019/2020. `https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2019-2020.pdf`, 2020.

[4] Mitchell, T. M. Machine learning. *New York*, 1997.

[5] Friedman, J.; Hastie, T.; et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[6] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[7] Gunetti, D.; Picardi, C. Keystroke analysis of free text. *ACM Transactions on Information and System Security (TISSEC)*, volume 8, no. 3, 2005: pp. 312–347.

[8] Latto, N. What is a sniffer and how can you prevent sniffing? Accessed: Feb. 28, 2021. [online], Available: `https://www.avg.com/en/signal/what-is-sniffer`.

[9] Szor, P. *The Art of Computer Virus Research and Defense*. Addison Wesley Professional, 2005.

[10] Egele, M.; Scholte, T.; et al. A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, volume 44, no. 2, 2012: p. 6.

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

[12] Alrabaee, S.; Shirani, P.; et al. On the feasibility of malware authorship attribution. In *International Symposium on Foundations and Practice of Security*, Springer, 2016, pp. 256–272.

[13] Or-Meir, O.; Nissim, N.; et al. Dynamic malware analysis in the modern eraA state of the art survey. *ACM Computing Surveys (CSUR)*, volume 52, no. 5, 2019: p. 88.

[14] Moser, A.; Kruegel, C.; et al. Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, IEEE, 2007, pp. 421–430.

[15] Damodaran, A.; Di Troia, F.; et al. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, volume 13, no. 1, 2017: pp. 1–12.

[16] Saad, S.; Briguglio, W.; et al. 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*, INSTICC, SciTePress, 2019, pp. 528–535.

[17] Vinayakumar, R.; Alazab, M.; et al. Robust intelligent malware detection using deep learning. *IEEE Access*, volume 7, 2019: pp. 46717–46738.

[18] Kumar, N.; Mukhopadhyay, S.; et al. Malware classification using early stage behavioral analysis. In *2019 14th Asia Joint Conference on Information Security (AsiaJCIS)*, IEEE, 2019, pp. 16–23.

[19] Han, W.; Xue, J.; et al. MalInsight: a systematic profiling based malware detection framework. *Journal of Network and Computer Applications*, volume 125, 2019: pp. 236–250.

[20] Kephart, J. O.; Arnold, W. C. Automatic extraction of computer virus signatures. In *4th virus bulletin international conference*, 1994, pp. 178–184.

[21] Webb, A. R. *Statistical pattern recognition*. John Wiley & Sons, 2003.

[22] Raff, E.; Nicholas, C. A Survey of Machine Learning Methods and Challenges for Windows Malware Classification. *arXiv preprint arXiv:2006.09271*, 2020.

[23] Ye, Y.; Li, T.; et al. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)*, volume 50, no. 3, 2017: pp. 1–40.

[24] Ucci, D.; Aniello, L.; et al. Survey of machine learning techniques for malware analysis. *Computers & Security*, volume 81, 2019: pp. 123–147.

[25] Raff, E.; Zak, R.; et al. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques*, volume 14, no. 1, 2018: pp. 1–20.

[26] Santos, I.; Brezo, F.; et al. Idea: Opcode-sequence-based malware detection. In *International Symposium on Engineering Secure Software and Systems*, Springer, 2010, pp. 35–43.

[27] Bilar, D. Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics*, volume 1, no. 2, 2007: pp. 156–168.

[28] Siddiqui, M.; Wang, M. C.; et al. Data mining methods for malware detection using instruction sequences. In *Artificial Intelligence and Applications*, 2008, pp. 358–363.

[29] Microsoft. PE Format - Win32 apps. Accessed: Feb. 28, 2021. [online], Available: `https://docs.microsoft.com/en-us/windows/win32/debug/pe-format`.

[30] Gibert, D.; Mateu, C.; et al. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *Journal of Network and Computer Applications*, volume 153, 2020: p. 102526.

[31] Aslan, Ö. A.; Samet, R. A comprehensive review on malware detection approaches. *IEEE Access*, volume 8, 2020: pp. 6249–6271.

[32] Schultz, M. G.; Eskin, E.; et al. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, IEEE, 2001, pp. 38–49.

[33] Cohen, W. W. Learning Trees and Rules with Set-valued Features. In *AAAI/IAAI, Vol. 1*, 1996, pp. 709–716.

[34] Kolter, J. Z.; Maloof, M. A. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, volume 7, no. Dec, 2006: pp. 2721–2744.

[35] Wadkar, M.; Di Troia, F.; et al. Detecting malware evolution using support vector machines. *Expert Systems with Applications*, volume 143, 2020: p. 113022.

[36] Yang, L.; Liu, J. TuningMalconv: malware detection with not just raw bytes. *IEEE Access*, volume 8, 2020: pp. 140915–140922.

[37] Gao, X.; Hu, C.; et al. Malware classification for the cloud via semi-supervised transfer learning. *Journal of Information Security and Applications*, volume 55, 2020: p. 102661.

[38] Xue, D.; Li, J.; et al. Malware classification using probability scoring and machine learning. *IEEE Access*, volume 7, 2019: pp. 91641–91656.

[39] Zhong, W.; Gu, F. A multi-level deep learning system for malware detection. *Expert Systems with Applications*, volume 133, 2019: pp. 151–162.

[40] Raff, E.; Sylvester, J.; et al. Learning the pe header, malware detection with minimal domain knowledge. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 121–132.

[41] Kolosnjaji, B.; Eraisha, G.; et al. Empowering convolutional networks for malware classification and analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 3838–3845.

[42] Singh, T.; Di Troia, F.; et al. Support vector machines and malware detection. *Journal of Computer Virology and Hacking Techniques*, 2015: pp. 1–10.

[43] Asquith, M. Extremely scalable storage and clustering of malware metadata. *Journal of Computer Virology and Hacking Techniques*, volume 12, no. 2, 2016: pp. 49–58.

[44] Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, IEEE, 2015, pp. 11–20.

[45] Baldangombo, U.; Jambaljav, N.; et al. A static malware detection system using data mining methods. *arXiv preprint arXiv:1308.2831*, 2013.

[46] Hall, M.; Frank, E.; et al. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, volume 11, no. 1, 2009: pp. 10–18.

[47] Shafiq, M. Z.; Tabish, S. M.; et al. Pe-miner: Mining structural information to detect malicious executables in realtime. In *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2009, pp. 121–141.

[48] Merkel, R.; Hoppe, T.; et al. Statistical detection of malicious PE-executables for fast offline analysis. In *IFIP International Conference on Communications and Multimedia Security*, Springer, 2010, pp. 93–105.

[49] Ye, Y.; Wang, D.; et al. IMDS: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 1043–1047.

[50] Basole, S.; Di Troia, F.; et al. Multifamily malware models. *Journal of Computer Virology and Hacking Techniques*, 2020: pp. 1–14.

[51] Mohaisen, A.; Alrawi, O.; et al. Amal: High-fidelity, behavior-based automated malware analysis and classification. *computers & security*, volume 52, 2015: pp. 251–266.

[52] Ahmadi, M.; Ulyanov, D.; et al. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the sixth ACM conference on data and application security and privacy*, 2016, pp. 183–194.

[53] Islam, R.; Tian, R.; et al. Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, volume 36, no. 2, 2013: pp. 646–656.

[54] Lakhotia, A.; Walenstein, A.; et al. Vilo: a rapid learning nearest-neighbor classifier for malware triage. *Journal of Computer Virology and Hacking Techniques*, volume 9, no. 3, 2013: pp. 109–123.

[55] Stanfill, C.; Waltz, D. Toward memory-based reasoning. *Communications of the ACM*, volume 29, no. 12, 1986: pp. 1213–1228.

[56] Wilson, D. R.; Martinez, T. R. Improved heterogeneous distance functions. *Journal of artificial intelligence research*, volume 6, 1997: pp. 1–34.

[57] Fix, E.; Hodges Jr, J. L. Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, DTIC Document, 1951.

[58] Dudani, S. A. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, volume SMC-6, no. 4, 1976: pp. 325–327.

[59] Quinlan, J. R. Induction of decision trees. *Machine learning*, volume 1, no. 1, 1986: pp. 81–106.

[60] Eberhart, R.; Kennedy, J. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, Citeseer, 1995, pp. 1942–1948, doi:10.1109/ICNN.1995.488968.

[61] Kuhn, M.; Johnson, K.; et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[62] Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation*, volume 9, no. 8, 1997: pp. 1735–1780.

[63] Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*, volume 18, no. 5-6, 2005: pp. 602–610.

[64] Vincent, P.; Larochelle, H.; et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, volume 11, no. 12, 2010.

[65] Carrera, E. Pefile. `https://github.com/erocarrera/pefile`, 2017.

[66] Scikit-learn. scikit-learn.org. `https://scikit-learn.org/`, 2021.

[67] Wettschereck, D.; Aha, D. W.; et al. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, volume 11, no. 1-5, 1997: pp. 273–314, doi:10.1023/A:1006593614256.

[68] Suárez, J. L.; García, S.; et al. A tutorial on distance metric learning: Mathematical foundations, algorithms, experimental analysis, prospects and challenges. *Neurocomputing*, 2020, to be published.

[69] Weinberger, K. Q.; Blitzer, J.; et al. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, 2006, pp. 1473–1480.

[70] Goldberger, J.; Hinton, G. E.; et al. Neighbourhood components analysis. In *Advances in neural information processing systems*, 2005, pp. 513–520.

[71] Weinberger, K. Q.; Tesauro, G. Metric learning for kernel regression. In *Artificial Intelligence and Statistics*, 2007, pp. 612–619.

[72] Guyon, I.; Weston, J.; et al. Gene selection for cancer classification using support vector machines. *Machine learning*, volume 46, no. 1, 2002: pp. 389–422.

[73] Kozák, M. *Static Malware Detection using Recurrent Neural Networks*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, 2020.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] Jureček, M., Lórencz, R. Malware Detection Using a Heterogeneous Distance Function. In: *Computing and Informatics*. Volume 37, no. 3, pp. 759-780, 2018.

The paper has been cited in:

- Wang, Z., Han, W., Lu, Y., & Xue, J. A Malware Classification Method Based on the Capsule Network. In: *International Conference on Machine Learning for Cyber Security*, pp. 35-49, 2020. *(citation indexed in Scopus and Google Scholar)*

[A.2] Jureček, M., Lórencz, R. Distance Metric Learning using Particle Swarm Optimization to Improve Static Malware Detection. In: *Proceedings of Int. Conf. on Information Systems Security and Privacy (ICISSP)*. pp. 725-732, Valletta, Malta, 2020.

[A.3] Jureček, M., Kozák, M. Representation of PE Files using LSTM Networks. In: *Proceedings of Int. Conf. on Information Systems Security and Privacy (ICISSP)*. pp. 516-525, Vienna / Virtual event, Austria, 2021.

[A.4] Jureček, M., Jurečková, O., Lórencz, R. Improving Classification of Malware Families using Learning a Distance Metric. In: *Proceedings of Int. Conf. on Information Systems Security and Privacy (ICISSP)*. pp. 643-652, Vienna / Virtual event, Austria, 2021.

[A.5] Jureček, M., Lórencz, R. Application of Distance Metric Learning to Automated Malware Detection. 2021, (the paper was submitted to the journal *IEEE Access*).

Remaining Publications of the Author Relevant to the Thesis

[A.6] Jureček, M. Automatic Malware Detection. Ph.D. Minimum Thesis, Faculty of Information Technology, Prague, Czech Republic, 2017.

[A.7] Jureček, M. Automatická detekcia malware. In: *Sborník Počítačové architektury a diagnostika (PAD)*, Doksy, Czech Republic, pp. 35-38, 2019.

Remaining Publications of the Author

[A.8] Jureček, M., Buček, J., Lórencz, R. Side-Channel Attack on the A5/1 Stream Cipher. In: *Proceedings of 22nd Euromicro Conference on Digital System Design (DSD)*. pp. 633-638, Kallithea, Greece, 2019.