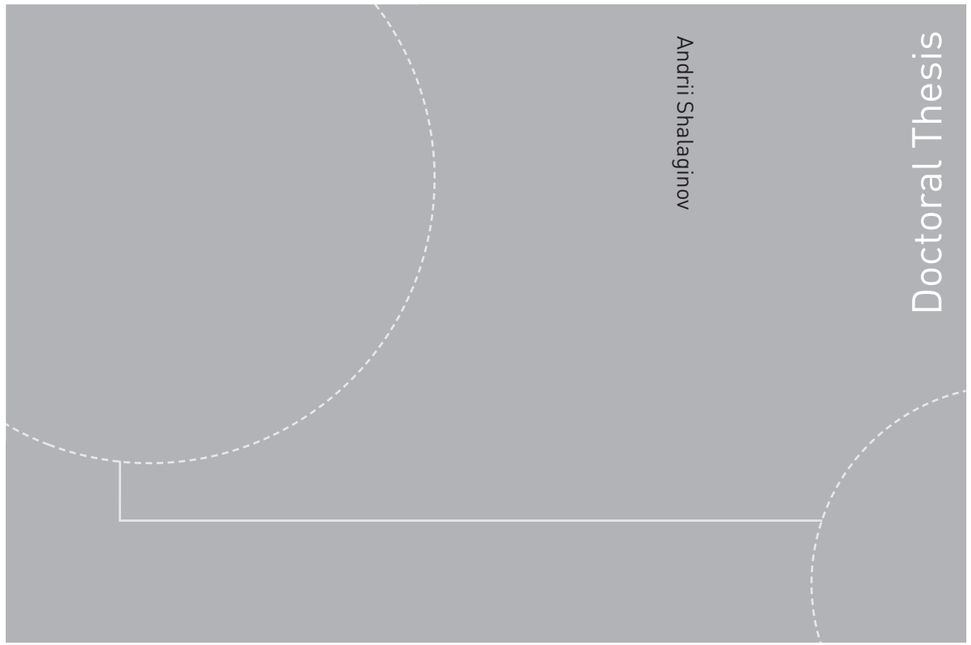
ISBN 978-82-326-2906-0 (printed version)
ISBN 978-82-326-2907-7 (electronic version)
ISSN 1503-8181



Doctoral theses at NTNU, 2018:57

Andrii Shalaginov

Advancing Neuro-Fuzzy Algorithm for Automated Classification in Largescale Forensic and Cybercrime Investigations

Adaptive Machine Learning for Big Data Forensic

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

NTNU
Norwegian University of Science and Technology

Doctoral theses at NTNU, 2018:57

NTNU

Andrii Shalaginov

Advancing Neuro-Fuzzy Algorithm for Automated Classification in Largescale Forensic and Cybercrime Investigations

Adaptive Machine Learning for Big Data Forensic

Thesis for the degree of Philosophiae Doctor

Gjøvik, February 2018

Norwegian University of Science and Technology
Faculty of Information Technology
and Electrical Engineering
Department of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology
and Electrical Engineering
Department of Information Security and Communication
Technology

ISBN 978-82-326-2906-0 (printed version)
ISBN 978-82-326-2907-7 (electronic version)
ISSN 1503-8181

Doctoral theses at NTNU, 2018:57

Printed by Skipnes Kommunikasjon as

Sammendrag

Etterforskere som arbeider med cyberkriminalitet blir utfordret av den store mengden av og kompleksiteten på digitale data som blir beslaglagt i kriminalsaker. Menneskelige eksperter er tilstede i retten og tar beslutninger basert på de digitale data og bevisene som er funnet. Det er derfor nødvendig å kombinere automatiske analyser med en representasjon av de digitale data og bevis som er forståelig for mennesker.

Maskinlæringsmetoder, som kunstige nevrale nettverk, støttevektormaskiner og bayesianske nettverk har blitt benyttet vellykket innenfor digital etterforsking. Utfordringene er at disse metodene verken gir modeller som er lett forståelig for mennesker, eller virker uten forkunnskap. Vår forskning er inspirert av det fremvoksende området computational forensics. Vi fokuserer på metoden neuro-fuzzy rule-extraction, en lovende hybrid intelligensmodell. Bidraget går til å forbedre ytelsen av neuro-fuzzy til å finne presise fuzzy- regler som er forståelige for mennesker. Disse reglene kan bli presentert og forklart i retten, noe som er bedre enn et sett med numeriske parametere tatt fra en mer abstrakt maskinlæringsmodell.

I starten av vår forskning på neuro-fuzzy metoden fant vi at dens anvendelse innenfor digital etterforskning var lovende, men med en del ulemper. Disse inkluderer (i) dårlig ytelse når det gjelder læring av modeller, fra den virkelige verden, sammenlignet med andre rådende metoder innenfor maskinlæring, (ii) en del av fuzzy-reglene er så store at ingen menneskelig ekspert kan forstå dem, (iii) en sterk overtilpasning av modeller, forårsaket av den store mengden fuzzy-regler, og (iv) en iboende læringsprosedyre som forsømmer deler av dataene og derfor blir unøyaktig. På bakgrunn av denne kritikken har neuro-fuzzy metodens latente potensiale ikke blitt mye benyttet innenfor dette området enda.

Bidragene fra dette verket er som følger: (1) teoretisk i forbedring av neuro-fuzzy

metoden og (2) empirisk gjennom eksperimentell design ved hjelp av storskala datasett fra domenet digital etterforskning. Hele studien ble utført 2013-2017 ved gruppen for digital etterforskning ved NTNU.

Add. 1. Vi har revidert neuro-fuzzy metoden, og derfor først bidratt innenfor maskinlæringsdomenet og dernest til anvendelsen innenfor storskala digital etterforskning. Spesielt, (i) har vi foreslått utforskende dataanalyser for å forbedre initialisering av selvorganiserende kart og generalisering av neuro-fuzzy metoden rettet mot storskala datasett; (ii) vi har også forbedret kompaktheten og generaliseringen til fuzzy-patches, noe som resulterte i økt nøyaktighet og robusthet av metoden ved hjelp av chi-kvadrat godhet av passformtest; (iii) vi laget en ny medlemskapsfunksjon basert på gaussisk multinomisk fordeling som tar høyde for representasjonen av fuzzy-patches som en statistisk estimert hyperellipsoide; (iv) vi reformulerte anvendelsen av neuro-fuzzy til å løse multiklasseproblemer i stedet for konvensjonelle toklasseproblemer; (v) tilslutt designet vi en ny fremgangsmåte for å modellere ikke-lineære data ved hjelp av deep learning og neuro-fuzzy, som resulterte i en deep neuro-fuzzy arkitektur.

Add. 2. Den eksperimentelle studien inkluderer bred evaluering av de foreslåtte forbedringene med hensyn til de utfordringene og kravene fra den varierte anvendelsen fra den reelle verden, inkludert: (i) rådende datasett, som Android malware datasettet, detektering av nettverksinnbrudd i KDD CUP 1999 og datasettet med brannmurer for web-applikasjoner, PKDD 2007. I tillegg ble det brukt andre datasett som er akseptert i miljøet, inkludert storskala datasett som SUSY og HIGGs. (ii) I tillegg ble det gjort en ny storskala innsamling av Windows Portable Executable 32-bit skadevare filer som en del av dette PhD-arbeidet. Det består av 328,000 merkede prøver av skadevare som representerer 10,362 familier og 35 kategorier; disse ble videre testet som ikke-trivielle multiklasseproblemer som ikke var tilstrekkelig studert i litteraturen eller utforsket tidligere.

Abstract

Cyber Crime Investigators are challenged by the huge amount and complexity of digital data seized in criminal cases. Human experts are present in the Court of Law and make decisions with respect to the digital data and evidence found. Therefore, it is necessary to combine automated analysis and human-understandable representation of digital data and evidences.

Machine Learning methods such as Artificial Neural Networks, Support Vector Machines and Bayes Networks have been successfully applied in Digital Investigation & Forensics. The challenge however is in the fact that these methods neither provide precise human-explainable models nor can work without prior knowledge. Our research is inspired by the emerging area of Computational Forensics. We focus on the Neuro-Fuzzy rule-extraction classification method, a promising Hybrid Intelligence model. The contribution goes towards the improved performance of Neuro-Fuzzy in extracting accurate fuzzy rules that are human-explainable. These rules can be presented and explained in a Court of Law, which is better than a set of numerical parameters obtained from more abstract Machine Learning models.

In our initial research on the Neuro-Fuzzy method, we found that its application in Digital Forensics was promising, but with a number of drawbacks. These include (i) poor performance in learning from real-world in comparison to other state of the art Machine Learning methods, (ii) a number of output fuzzy rules so large that no human expert can understand them, (iii) a strong model overfitting caused by the huge number of fuzzy rules, and (iv) an intrinsic learning procedure that neglects part of the data, which therefore becomes inaccurate. Due to this criticism, Neuro-Fuzzy method's latent potential has not been widely applied to the area yet.

The contribution of this work is the following: (1) theoretical in the improvement of Neuro-Fuzzy method and (2) empirical in the experimental design using large-

scale datasets in Digital Forensics domain. The entire study was conducted during 2013-2017 at the NTNU Digital Forensics Group.

Add. 1. Neuro-Fuzzy was revised and therefore we first contributed to the Machine Learning domain and subsequently the large-scale Digital Forensics application. In particular, (i) we proposed exploratory data analysis to improve Self-Organizing Map initialization and generalization of the Neuro-Fuzzy method targeting large-scale datasets; (ii) we also improved the compactness and generalization of fuzzy patches, resulting in the increased accuracy and robustness of the method through a chi-square goodness of fit test; (iii) we constructed the new membership function based on Gaussian multinomial distribution that considers fuzzy patches representation as a statistically estimated hyperellipsoid; (iv) we reformulated the application of the Neuro-Fuzzy in solving multi-class problems rather than conventional two classes problems; (v) finally, we designed a new approach to model non-linear data using Deep Learning and Neuro-Fuzzy method that results in a Deep Neuro-Fuzzy architecture.

Add. 2. The experimental study includes extended evaluation of the proposed improvements with respect to the challenges and requirements of a variety of different real-world applications, including: (i) state of the art datasets like the Android malware dataset, network intrusion detection KDD CUP 1999 and web application firewalls PKDD 2007 datasets. Moreover, community-accepted datasets from UCI collection were also used, including large-scale datasets such as SUSY and HIGGS. (ii) A new, novel large-scale collection of Windows Portable Executable 32-bit malware files was also composed as a part of this PhD work. It consists of 328,000 labelled malware samples that represent 10,362 families and 35 categories; these were further tested as non-trivial multi-class problems, neither sufficiently studied in the literature nor previously explored.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of philosophiae doctor (PhD) at the Norwegian University of Science and Technology (NTNU). The work has been performed at the Department of Information Security and Communication Technology, Faculty of Information Technology and Electrical Engineering at the Norwegian University of Science and Technology from 2013 until 2017. The research was carried under the supervision of Professor Katrin Franke, Professor Slobodan Petrović and Professor Mario Köppen.

I feel grateful to study and be a part of the Department. I had great opportunities and achieved many challenging goals during these four years. High level of technical and administrative support played a crucial role in conducting this research. I gratefully acknowledge the financial support from Department, Research School of Computer and Information Security and a travel award granted by Journal Artificial Intelligence.

I would like to express my thanks to Karl Hiramoto from VirusTotal for support and academic access to the anti-virus databases. That made it possible to make a contribution to the area and to compose a novel labelled multi-class Windows malware dataset.

Acknowledgements

I would like to gratefully acknowledge my advisors Prof. Dr. Katrin Franke, Prof. Dr. Slobodan Petrović and Prof. Dr. Mario Köppen for their fruitful discussions and valuable guidance during these years. Thank you for all your support and important advices regarding my work. I am thankful to Katrin for all the motivation, inspiration, vision, practical advices and mentoring, which made a substantial contribution to my professional and personal development. In addition, I would like to thank to members of the evaluation committee, Prof. Dr. Ajith Abraham, Prof. Dr. Magnus Almgren, Prof. Dr. Basel Katt and head of the committee Prof. Dr. Laura Georg, who agreed to review my PhD thesis and provide valuable comments.

I am thankful to the Department of Information Security and Communication Technology, Faculty of Information Technology and Electrical Engineering at the Norwegian University of Science and Technology for being able to carry out this research and creating a highly-productive environment. This research and proof-of-concept demonstrations would not be possible in a given time frame without provided advanced hardware capabilities and financial support from the Department and NTNU Digital Forensics Group.

A number of administrative staff and faculty members played an important role in this research. I would like to thank to Kathrine Huke Markengbakken, Jingjing Yang, Rachael McCallum, Florissa Abreu, Ingrid von Schantz Bakka, Urszula Nowostawska, Hilde Bakke, Maria Henningsson, Jan Kåre Testad, Per David Nielsen and Anne Aandalen who supported and gave important advices on different stages of my PhD research. Special thanks to the Head of the Department Nils Kalstad Svendsen and also to Laura Georg, Sofie Nystrøm and Morten Irgens. My academic experience benefited from discussions with senior faculty members, Geir Olav Dyrkolbotn, Thomas Kemmerich, Stewart James Kowalski, Patrick Bours,

Basel Katt, Mariusz Nowostawski, Stefan Axelsson and Stephen Wolthusen.

I have greatly benefited from collaboration and discussions at Kripos and Økokrim. In addition, I am thankful to COINS Research School of Computer and Information Security and in particular Hanno Langweg for organizing all the seminars, winter and summer schools during these four years. It was a great time and irreplaceable contribution to my personal development, career growth and valuable networking.

A number of researchers that played a direct role in the research and to whom I am thankful for exciting collaboration and achievements: Gaute Wangen, Lars Strande Grini, Ali Dehghantanha, Edgar Lopez, Christoffer V. Hallstensen, Sergii Banin and Carl Stuart Leichter. Moreover, I am grateful to my fellow colleagues for their ideas and time spent together, Ambika Shrestha Chitrakar, Kiran Bylappa Raja, Vivek Agrawal, Vasileios Gkioulos, Martin Stokkenes, Guoqiang Li, Goitom Kahsay Weldehawaryat, Martin Aastrup Olsen, Dmytro Piatkivskyi, Anastasiia Moldavska, Oleksandr Semeniuta, Ivanna Baturynska, Kyle Andrew Porter, Ctirad Sousedik, Jan William Johnsen, Yi-Ching Liao. Additional thanks to all my friends and people in my life for being there for me and playing an important role in my personal life.

At the end I am grateful to my wife Marina and my parents for their patience, generous support and love that made this PhD possible and meaningful.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This chapter is devoted to the scope of the dissertation. In particular, Section 1.1 presents objectives and motivations behind this research, specifically why Computational Intelligence is important in the field of Digital Forensics. Next, Section 1.2 gives a brief insight into the current state of the art along with challenges that arose in data analysis. These challenges are addressed in corresponding research questions with relevant contributions of the thesis explained in the Section 1.3. Finally, the outline and the structure are given in the Section 1.5.

1.1 Motivation & Objectives

Forensic Science is an emerging area consisting of the application of different methodological approaches in Crime investigation [245]. Such utilization requires constant improvement upon previous methods in an agile environment exploited by perpetrators. Digital Forensics is one of the major sub-fields focused on revealing evidence found on digital data carriers and within ICT infrastructure [63]. Examination of found information for further representation in a Court of Law has been largely based on manual search, pattern matching, and an analysis of found traces. However, this comes to looking for a "needle in a haystack", which might be infeasible despite a knowledgeable manual analysis of found evidence. As a result, Digital Forensics is in need of automated data analytics and processing for Decision Support.

The Big Data paradigm became inevitable in every aspect of modern digital life. Garfinkel [159] wrote that an average computer's HDD of a size 2TB requires more than 7 hours to image a device alone, without even mentioning analysis or file carving. And this not taking into account mid-class servers with a storage

space from 10-20TB. On the other hand, mobile phones became extremely popular and have been converting from simple end-user communication terminals into a powerful and resourceful tools capable of massive parallel computing and storage of a variety of log data from various sources, including GPS and a number of other sensors beside user's photos and documents [239]. By the end of 2016, Apple's Iphone 7 or Google's Pixel became able to store up to 128GB [126] of personal sensitive information that could be targeted by adversaries. Naturally, palm-sized devices were hit by a number of malware, including botnets and spyware [270]. Malware (or malicious software) are software that perform unwanted actions in a targeted system. McAfee malware zoo [271] included 440,000,000 samples by Q2 2015. Malware poses a significant threat to every device connected to the Internet in terms of privacy and economic loss. The majority target the MS Windows NT Operation Systems (OS) family that has been in use since the end of the 1990s. In addition to this, the threat landscape rapidly changes under the BYOD policy implemented by companies around the globe. Overall, the amount of data is enormous, which complicates the work of a forensics analyst if ICT is a tool used to commit a crime or has become the target of a crime committed.

The challenges of Big Data in Digital Forensics has been there for over decade. For example, the Enron case back in 2001 [228] shows how the investigation recovered 619,446 emails resulting in 160GB of data, partially plain text. Traditional Artificial Intelligence methods (such as Support Vector Machines, Naive Bayes Classifier and K-Nearest Neighbor Classifier) are simply incapable of handling such data or producing meaningful traces of evidence. Another example is the December 2016 discovery of Yahoo!'s second breach, resulting in the leak of a billion accounts' worth of user's personal data [148]. With millions of users accessing their services every day, it quickly becomes impossible to find whatever malicious actions reside in the Yahoo! server logs. In their report, Ernst & Young [140] revealed that Big Data is no longer necessarily an insurmountable obstacle anymore in the field, especially when considering 5Vs [266]: volume, velocity, variety, veracity and value. At this point distributed techniques for data storage are capable of handling the *volume* and the *velocity* of newly generated data. Additionally, the *veracity* and *variety* can be handled by Machine Learning to be able to extract corresponding *value*. The goal therefore becomes to produce forensically-sound evidence that can be presented further in a Court of Law as depicted in the Figure 1.1.

To tackle the aforementioned challenges, a strong need for automated approaches arises. Manual analysis is no longer considered a reasonable approach, considering the number of pieces of data that need to be processed. Since the 1950's, ARTIFICIAL INTELLIGENCE has become a popular scientific field of study. One of the

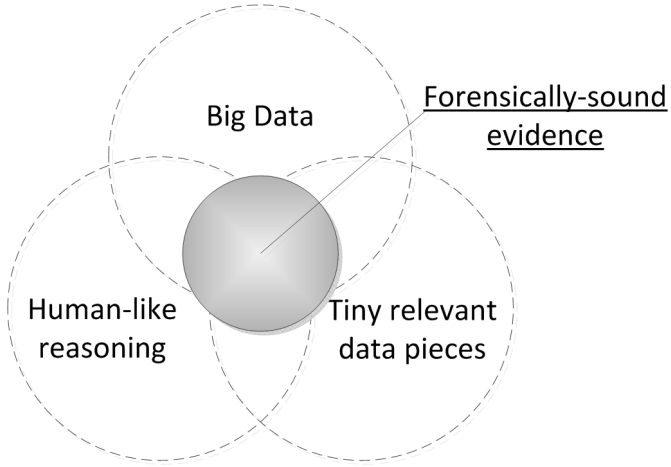


Figure 1.1: A challenge in modern Digital Forensics

main approaches in AI research is COMPUTATIONAL INTELLIGENCE, which uses nature-inspired methods to target a real-world problem and derive understandable reasoning. Put simply, Machine Learning has shown its effectiveness in Information Security before [60, 334, 398]. As results, these methods can be applied to study the materials from crime scenes. Such an area of study is called COMPUTATIONAL FORENSICS and covers the application of computer-based methods to Crime Investigation. Computational Intelligence is closely related to so-called SOFT COMPUTING, a synergy of imprecision tolerance and model robustness that are of value to Crime Investigation due to the chaotic environment and missing pieces of information, as presented in the Figure 1.2. Contrary to the conventional HARD COMPUTING that requires a crisp answer to a defined problem, SC derives an inexact solution as described by Zadeh in 1994 [455]. Therefore, it leaves a decision up to the forensics expert's judgement.

Considering the acute need for automated data processing, Neuro-Fuzzy rule-extraction classification methods emerge as one of the most prominent SC methods, a synergy of human-like linguistic rule-based Fuzzy Logic and brain-inspired Neural Networks modelling. Separately, Fuzzy Logic requires thorough manual tuning of the model's parameters, while Neural Networks produce a rather complex and hardly presentable weights-based model. The Neuro-Fuzzy method has not yet been sufficiently studied as a methodological approach for Crime Investigation. The generic Neuro-Fuzzy performs poorly on problems related to Digital Forensic data analytics. Considering the "No free lunch theorem" by Wolpert et al. [442] however, it appears that this method is capable of providing an optimal trade-off between accuracy, computational complexity and interpretability of the

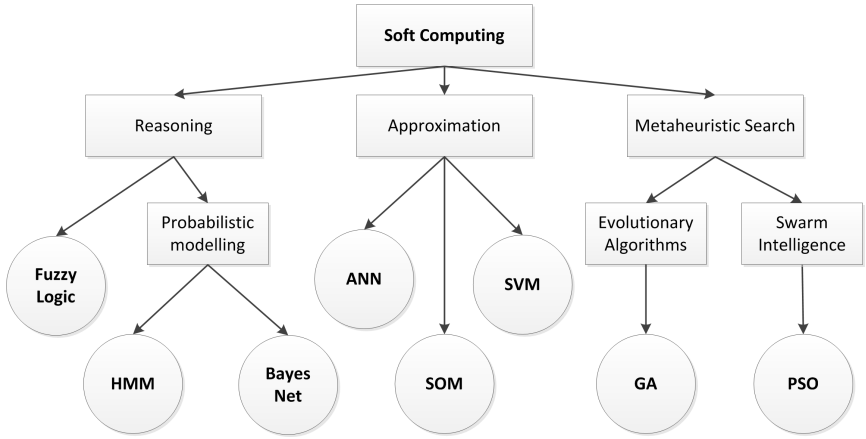


Figure 1.2: Overview of Soft Computing methods

derived model.

1.2 Related Works & Challenges

Computational Criminology can use SC methods as a Computational Intelligence approach to facilitate the Investigation process as stated by Franke et al. [151]. There have been several previously identified areas of forensics from which different Computational Intelligence approaches can be drawn to perform forensically-sound data analysis, as it can fulfill the Dauber Standards [5]. In this thesis, we consider Soft Computing as the most promising however, since the case data may not have an ideal match with previous cases or known criminal patterns. Zadeh [455] that in contrast to hard computing can be employed SC in his 1994 work to unify the decision making process together with the human cognitive process. The main difference from conventional hard computing is the flexibility in analytical model construction that does not require precisely stated parameters and characteristics. From the literature, we can see that significant research has been done in the area of Cyber Crime Investigation, for example [248]. One of the main advantages of SC is that it has output terms such as likelihood, probability, proximity score and so on. This gives flexibility to an analyst in making decisions, since the result is not crisp and can be corrected considering other factors. Thus, the rigorous answers are not necessarily needed, since the decision will be ultimately made the human brain.

Furthermore, Zadeh and Dickerson et al. [120] proposed SC concept Neuro-Fuzzy that is a synergy of Fuzzy Logic (FL) and Artificial Neural Network (ANN) with two stages: rough, unsupervised placement of so-called fuzzy patches using Self-

Organizing Map (SOM), and tuning of the fuzzy rules using ANN. One of the major challenges of the first stage of this approach is to define the size of the map beforehand, either by growing it as proposed by Alahakoon et al. [50] in 1998, or else by applying the "rule of thumb" as elaborated by Vesanto et al. [424] in 2000. Regrettably, the first method demands enormous computational resources, while the second requires an extreme number of SOM grid nodes when dealing with large datasets. Landress [238] highlighted this peculiarity of SOM, especially when dealing with unsupervised learning in Intrusion Detection. The application of both methods results in an overfitted model with a high number of fuzzy rules—too unreliable for presentation in a Court of Law. According to Kosko [233], one uses elliptic regions for better describing the data in each SOM node, yet there are no qualitative metrics on how to find the pseudo-radius of this hyperellipsoid. The typical solution is to define this number empirically and apply it to all fuzzy regions extracted using SOM, but this can result in major errors. Finally, the Membership Function (MF) construction used in the second stage of NF challenges processing since, according to Kosko [233], the projections of the hyperellipsoid are also used to construct the corresponding triangular MF. This does not guarantee however that the MF will incorporate mutual correlation between features determined by means of the stretchiness and angle of inclination of the ellipsoid. In [171], Guillaume studied various hybrid models and stated that NF is one of the most useful data approximation techniques. As written previously, the Neuro-Fuzzy method was used for Network Forensics according to Anaya et al. [59] to detect suspicious flows based only on TCP/IP LANs that have been compromised. NF method has been neither sufficiently explored nor crafted to be best used for different applications in Digital Forensics.

1.3 Scope & Research Questions

This project pursues multiple research objectives, including the generation of new knowledge in the field, improvement of existing algorithms, and collection of relevant large-scale data for proper testing. Therefore, the following general research questions were formulated:

- *Q1: Which Soft Computing algorithms are applicable in forensics data sciences and allow one to derive forensically sound intelligent decisions from the data without any structure or existing meta data, with respect to privacy issues and data protection?*

  Cybercrime Investigation is facing multiple challenges in analysing data from criminal cases due to the uniqueness of data and specific environment where data is stored. Therefore, it is necessary to highlight the areas where *Soft Computing* can find a successful application in contrast to classical com-

puter forensics methods. There already exist several solutions for storing and mining large-scale data, however, they may be infeasible for finding relevant information. From the other side, ML works fine, but results are hardly explainable and provide no way for one to find why the model was constructed in any specific form. We performed an overview of the relevant aspects of SC application with respect to key phases of Digital Forensics process. Moreover, we considered the fact that SC must comply with Daubert Standards [5] to be able to demonstrate sufficient proof of evidence in a Court of Law. The contribution and preliminary studies are published in [367, 371].

- *Q2: How does hybridization improve stand-alone Soft Computing algorithms to achieve admissibility and performance of evidence extraction for Digital Forensics applications?*

  The vital task in Digital Forensics Process is not just to preserve data for future analysis, yet also to extract meaningful evidence. So, the assumption is to use several approaches of Soft Computing to create reliable and fast hybrid intelligence solutions. First, from previous studies we found that the Neuro-Fuzzy method is one of the most promising models based on fuzzy rules that can derive human-explainable solutions. However, NF was not originally intended to be used for such purposes due to the low accuracy and high complexity of the model, while stand-alone usage of FL and ANN was simply inappropriate due to the challenges described. Second, we analysed Neuro-Fuzzy proposed by Kosko [233] and proposed an improvement by using exploratory data analysis through the Pearson correlation coefficient for better learning of SOM on the 1^{st} stage of NF. This was used instead of the "rule of thumb" and Vesanto method to achieve a higher degree of interpretability and an agreeable trade-off between complexity and accuracy. Finally, we were able to achieve much higher accuracy of the data described by significantly lower number of rules. The proposed method was tested on a number of different datasets of different dimensionality and complexity, including Android malware samples collection. The approach and achieved results were given in the [374, 375].

- *Q3: How Big Data analysis using Soft Computing can be optimized with respect to resources and time consumption while applying multi-objective mathematical optimization and high-performance computing?*

  To answer this question, we looked into algorithms from the perspective of the data processing demands. This elicited ways to apply the numerical optimization while preserving accuracy and improving response time. Despite the successful usage of parallel optimization and GPU, data analysis may

fail if the processing model is too complex or has nonlinear dependency on the amount. This is the case with the method proposed by Kosko. First, this method suggests the use of large amount of rectangular fuzzy patches Π that are intrinsically erroneous due to insufficient transition of data properties. To mitigate this, we suggest using elliptic fuzzy patches to have better goodness of fit to real world data. Second, the ellipsoid radius α is empirically defined, which requires additional efforts by the data analyst to tune it. Instead, more naive determination of the pseudo-radius α through χ^2-square test of goodness of fit results in a better data characterization as well as the elimination of mistaken data. Through these improvements we aim to achieve fast learning in soft computing even while dealing with Big Data. Also, we looked at ways that on-line learning can be improved in Neural Networks to facilitate data streams mining in Information Security. Moreover, it is important to keep in mind that it is more computationally efficient to perform information fusion in a dynamic environment using Neuro-Fuzzy. The approaches are suitable for small data problems, but no more reliable for processing vast amount of data. The method is described in the publication [373, 374].

- *Q4: How can data be better incorporated in the Neuro-Fuzzy rules-extraction classification method while using a lower number of more compact and better located fuzzy patches?*

  Membership functions define how well the degree of truth is transferred from data to new unknown samples. The original triangular or projection-based membership function proposed by Kosko cannot incorporate all data. On the contrary, transferring parameters of fuzzy patches to Gaussian MF allows one to provide robust estimation of membership degree with respect to data stretchiness and angle of rotation of fuzzy patches. So to mitigate the aforementioned challenge, we proposed a new membership function based on the hyper-ellipsoid parameters to incorporate all the variables from multinomial Gaussian distribution. Gaussian approximation offers a better degree of goodness of fit to the real-world data. By applying the suggested function and parallel optimization, we were able to achieve not only better performance, but also a significantly reduced number of fuzzy rules tuning iterations on the 2^{nd} stage of NF. The improvements were presented in the papers [373, 375].

- *Q5: Can Digital Forensics criminal cases with large data quantities be managed by Soft Computing models where fast and reliable response is required?*

  The CyberCrime Investigation is not only about post-mortem analysis of the log files, traces and system artifices found on digital data carriers, yet

also about proactive crime prevention. For example, Internet evidence collection requires interactive social media profiling with on-line adaptation of the statistical model. The general Soft Computing framework for cybercrime investigation application needs not just human-understandable model, but also an ability to be re-trained quickly while processing data travelling at high speed. Most Intrusion Detection Systems offer signature-based detection of suspicious activity, which can be inefficient in the detection of zero-day attacks. Therefore, we improved Neuro-Fuzzy to be able to facilitate Network Forensics Readiness by applying similarity-based detection. In addition to this, we investigated how Fuzzy Logic can be used for Information Security Risk Management, which has an inevitable overlap with Digital Forensics Readiness. By training from million-sample datasets such as KDD Cup 1999, the proposed model is capable of nearly real-time packet processing, suitable for modern networks. An improvement of the methods was described with corresponding use case analysis in contributions [372, 375, 376, 377].

- *Q6: What are ways to improve the generalization and performance of the Neuro-Fuzzy rule-extraction classification method for large-scale multinomial problems?*

  Computer Crime Investigation introduces a number of data analysis problems related to so-called multinomial classification problem. Contrary to conventional binary classification (*benign* vs. *malicious*), multinomial considers many sub-types of *malicious*. A particular subdivision is the detection of attacks in web firewalls such as described by dataset PKDD 2007; another is malware classification. A majority of researches consider only binary classification, yet this is neither sufficient nor relevant for modern information security. A novel dataset containing modern Windows PE32 malware samples was used to show a prospective application of the automated multinomial malware detection. A number of malware categories and families emerged over last decade targeting Microsoft Windows, since it is the most attractive platform for virus developers. Static and dynamic analysis can reveal information relevant to classification characteristics in each malware category. To study this problem, we created a novel dataset of PE32 executables originally consisting of 400k malware samples. First, we proposed to use limits on a configuration of each SOM node clustering to be able to produce statistically-sound fuzzy rules. Second, as a way to enhance accuracy and generalization of NF, we proposed applying new single-output architecture of the model. Third, a new output defuzzification function was suggested that helped to improve the accuracy of the original NF method.

Finally, most of the ML approaches have considerably worse performance on multinational problems when dimensionality is large and data has non-linear properties. A novel Deep Neuro-Fuzzy rules-extraction classification approach was developed to mitigate a high level of non-linearity while giving robust classification by using a higher abstraction level. The dataset, analysis of results, and proposed improvements are contributed in the papers [167, 372, 377, 379].

Our goal was to integrate Soft Computing into the Digital Forensics process, more specifically by building decision support models as shown in the Figure 1.3. In addition to the different domains, the nature of the data being analysed was considered. The proposed improvements and optimization of the hybrid Neuro-Fuzzy methods gave a prospective approach on how to ensemble both the machine learning from data and the construction of human understandable models. To summarize, the contribution of the dissertation is an intelligent model that is capable of handling a variety of large-scale problems that were validated on community-accepted datasets, as well as newly constructed ones.

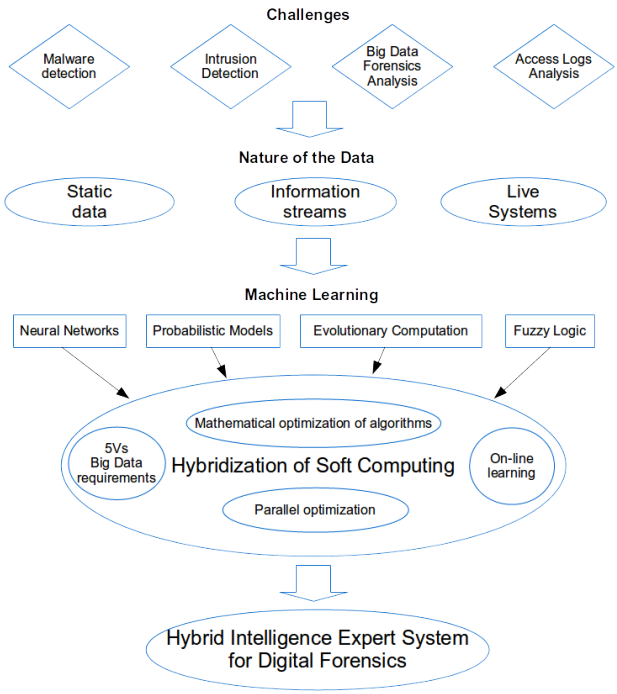


Figure 1.3: Contribution towards application of Soft Computing for Digital Forensics

1.4 Contributions

The thesis provides a theoretical background in the areas of Forensics Science and Cyber Crime Instigation. Foundations of Machine Learning, Soft and Hard Computing are followed by the state of the art in Fuzzy Logic, Artificial Neural Network and Neuro-Fuzzy. We focused on the in-depth study of Hybrid Intelligence with respect to the needs of Cyber Crime Investigation. Finally, challenges and limitations of current state of the art methods are given.

New computational methods that are designed to enhance the utility and performance of Neuro-Fuzzy approach. In particular, it presents an improvement to the 1^{st} NF step by using a new optimality metric for SOM size determination. Furthermore, a new fuzzy patches construction method based on the χ^2-test was proposed along with the corresponding membership function construction based on the Gaussian distribution of data in a multidimensional hyper-ellipsoid. Moreover, improvements targeting multi-class detection are given to be able to differentiate not only conventional two-classes problem, but also sub-classes. Finally, Deep Neuro-Fuzzy architecture supports the solution of such problems by introducing non-linearity components.

Finally, comprehensive overview of the proposed improvements with regard to application areas of Digital Forensics was given. We present the performance of our methods on different problems, including malware classification and network security. Another important contribution is the collected large-scale dataset covering modern Windows PE32 computer viruses labelled into categories and families.

1.5 Thesis Outline

The outline of the dissertation is given below:

Chapter 1 presents an overview of the research questions and challenges identified within this research work. The motivation and corresponding methodology is given along with a critical view of the related works in the area.

Chapter 2 includes a theoretical foundations in the areas of Digital Forensics, Machine Learning, Soft and Hard Computing are followed by the state of the art in Fuzzy Logic, Artificial Neural Network and Neuro-Fuzzy at the end of the chapter.

Chapter 3 shows new improved computational methods with corresponding justifications and analysis.

Chapter 4 contains practical results there were achieved while working on the methodology, including data collection, results analysis and study of applicability in real-world scenarios.

Chapter 5 Gives an overview of the contribution as a whole. The theoretical implications of the work are given along with the practical considerations of the application of Neuro-Fuzzy in Digital Forensics science. Finally, findings are summarized.

Chapter 2

State of the Art

This chapter gives a strong theoretical introduction into the topic that will serve as a further basis for building new knowledge and developing new methods. First, we focus on the overview of Forensics Science in general in the Section 2.1. Then, we consider Digital Forensics, its principles, challenges and ethical considerations in the Section 2.2. The idea is to give an overview of the area and sub-domains that can facilitated from an application of Soft Computing when analysing data seized from data carriers. Furthermore, we will introduce Machine Learning and statistics methods that can deal with data modelling and analysis in the Section 2.3. This includes conventional Hard and nature-inspired Soft Computing approaches. Finally, the Section 2.5 provides a literature overview and a basis for application of Hybrid Intelligence as a key factor for processing increased amounts of data as well as generation of forensically-sound evidence.

2.1 Forensic Science

Forensic Science is a general field of application of scientific methodologies for criminal investigations. Through application of such methods, one can discover evidence that may clarify the picture of a crime and establish links between a perpetrator and a victim. At this point, Forensic Science subdivides into many areas (also called Sciences) depending on the pieces of information and nature of evidence to be analysed. Below, we list the possible applications that are done in CF to facilitate the forensically-sound data analytic. Therefore, we selected the following forensics sub-fields:

- **Anthropology / Reconstruction** deals with the identification of a person in a legal setting, in particular through facial reconstruction. There are several

famous works performed in the last decade. Ibáñez et al. in 2009 [195] have studied an application of EC for craniofacial superimposition based on several cases. Another work has been done by Campomanes-Álvarez et al. [86] in 2013, where a similar problem was tackled using Fuzzy Logic in addition to Genetic Algorithm for an optimization of skull alignment done through search. Human identification by means of ears photos was proposed by De Tre et al. in 2014 [112] based on fuzzy set theory.

- **Biometrics** is an area that compounds the analysis of something that a person is. In the study done by Rughooputh et al. [343], a forensics application of ANN for the determination of the traces of the Raman images for fingerprints verification. Another application that was studied by Franke et al. in 2002 [152] is forensics handwriting verification used in banking sector and governmental organizations. They proposed to use regions on the image of a signature with further learning of a hybrid ANN and FL methods called Neuro-Fuzzy. The work produced high classification rate in addition to extracted linguistic rules.

- **Digital Forensics** covers a variety of methods whose scope are computer-related evidence. Several domains can be mentioned depending on the location and data to be analysed.

  - *Malware analysis* covers an examination of possible software applications with malicious intentions. According to the feasibility study by Singh et al. [394], most of the SC methods including ANN, FL, SVN can be applied in malware detection. Though there is a challenge to detecting unknown samples, SC methods still perform well when learning using a labelled dataset of known samples. The process of malware detection distinguishes two main areas of model application. First, static signatures are generated and detection is performed using artifacts that are found in the system. Second, the live system is monitored for behavioural patterns that malware can generate, including created and modified files, API calls, traffic, etc. For example, different ensembled methods based on NN and SVM were proposed by Veerwal et al.[114]. Shalaginov et al. [374] studied a trade-off between accuracy and interpretability of the hybrid NF model in 2015. The results showed that a smaller number of generalized FL rules results in better accuracy in mobile malware detection.
  - *Network Forensics* deals with the data in a transfer, that data in flow between computing nodes in an interconnected network. The investigation has developed a great interest in this area since a lot of cy-

bercrimes are committed using network means. Mukkamala et al. in 2003 [290] studied the features that matter for this type of forensics. The authors studied SVN and ANN and found that SVM overcomes ANN with respect to speed and scalability. Furthermore, in 2009 Liao et al. [249] presented techniques for expert system construction based on the FL. The expert systems showed great performance, for example in the detection of multiple attacks among normal traffic packets.

- **Social Network Mining** and analysis is a relatively new area based on the extraction of relevant information for crime investigation from corresponding social media. It includes the discovery of dangerous patterns, possible criminals and victims, etc. Lau et al. in 2014 [241] presented how Gibbs sampling methods can be used for social media mining. In particular, the number of Twitter messages from well-known criminals Anonymous group were analysed and classified.
- **Content identification** implies the approaches targeted at the detection and identification of the file types on the memory carriers. In his thesis [178], Harris stated how ANN can be applied in the identification of different file types, consisting of 5 image formats. Additionally, the author used up to 20,000 epochs to train the network; there was not much improvement however with respect to MSE[1]. TIFF format was the most effectively detected when using n consecutive bits from a file.
- **Mobile forensics analysis** became an inseparable part of Computer Forensics. The exponential growth of the number of mobile devices from early 2000 onward is only further complicated by emergence of computer-like smartphones. It became feasible to not only call and send text messages, but also use GPS locations, mobile Internet, store vast amount of private information, access bank services, etc. NIST has provided guidelines on mobile device forensics [206]. Among mobile OS, Android is the most popular platform considering open-source applicability and possibility to install 3rd party applications that may contain malicious payloads. Various Machine Learning methods can be used to differentiate between malicious and benign software [346, 371].
- **Network Intrusion Detection and Prevention** involves analysis and learning from the network traffic in order to detect illegal activities, information leakage, or anomalies. According to [42] by Abraham at al., the Evolutionary Algorithms were successfully tested as parts of IDS. They stated that such an approach can be used in developing

[1] MSE - mean square error

the automated system. Survey [193] gives an insight into the usage of Soft Computing methods in IDS. The authors stated that despite good accuracy and performance, there is still a demand to employ new and more advanced strategies in order to fight attacks. Furthermore, in [41], multiple paradigms including fuzzy rules and ensembled classifiers were used as data mining to construct intelligent IDS. The fuzzy classifier and genetic programming gave the best results in attacks detection. The fuzzy clustering method c-means was also applied in IDS [395] together with rough sets as feature selection methods. Traffic control systems are another field where SC found application. In the review [408], multiple approaches were named based on fuzzy techniques that make up one of the basic SC principles.

  - **Access control & Privacy Preserving** includes intelligent evaluation of the access to some resources. It can be an on-line learning that will give or deny access to some protected information and operations based on learning from human decisions or similar data. There have however not been many research papers on this topic. A physical access control based on the location is introduced in [182], where multiple RBF networks were used to denote each physical location, producing the location-aware engine.

- **Evidence Discovery & Surveillance** consists of the aggregation and characterization of important and relevant pieces of information out of chaotic and often agile environments. There can follow other important demands, such as uninterrupted mining of the information and new knowledge discovery and linking. In 2012 [307], researchers assembled a major collection of research articles that describe various SC techniques applicable in Surveillance Systems. Among them is the rough fuzzy method for image analysis and fuzzy rules.

- **Forensics Economics** targets unusual and illegal patterns in stored or transferred information by comparing "normal" patterns against questionable ones. Thang et al. [413] applied fuzzy inference engine was together with Neural Network to discover firms with fraud status. In this hybrid model, the membership functions of the features from finance reports and business information were used as inputs to construct fuzzy rules and make decisions, further employing the NN. According to authors, such a model is successful as a decision support system. Several types of credit card fraud were identified, and the application of neural network was demonstrated by Dukhi et al. [132]. They also suggested that the implementation of rules systems may help differentiate between specific types of fraud.

- **Forensics Identification** works to identify specific objects on a crime scene. Objects that can be identified or characterized range from DNA blood samples to documents. Stoffel [402] addressed a problem showing how accurate and understandable rules can be extracted from forensics data such as robberies and residential burglaries in the region of Lausanne, Switzerland. A Mamdani-type fuzzy inference system was used for the 70 features of the dataset. A work by De Vel et al. in 2001 [113] described how the e-mail content could be used for author identification. In particular, SVM was used over three authors to classify the 3 classes' e-mails with 21 features. Another work by Shrestha Chitrakar et al. in 2013 [101] depicted an overview of several ML methods, including Bayes Network and SVN, for author identification in emails. Both methods showed a strong performance for this task.

2.2 Cyber Crime Investigations

Digital Forensics, a sub-field of Forensic Science, covers the application of scientific methodology towards an analysis of crimes where information and communications systems (ICT) are involved. Digital Forensics however specializes only in the evidence present on the data carriers and computer-like devices that contain data storage.

Also, there will be an overview later on of the most common cases in Digital Forensics; since it is predominant in application of SC, we do not want it to shift the balance of this study. These include Network Forensics Readiness, malware analysis, etc.

2.2.1 Concepts in Digital Forensics

Digital Forensics has been heavily developed since the end of 1980's - beginning of 1990's, coming as a result of ICT system integration in many aspects of life. In 1934, Edmond Locard (1877-1966) suggested the so-called "exchange principle" defined by the assumption that there will be tiny traces of evidence found on crime scenes where in contact with a person or a thing [10]. Similarly, if a computer system has been involved in a crime as a tool or a target, it would most likely be reflected in a carrier. There exist a number of best practices and tools that the community utilizes in Crime Investigation and can be named de-facto standards. Below, we list those standards, organizations, and tools that have been accepted by the community during the last decade [75]. It is worth mentioning that most of the tools use keywords of full text search. We have not yet seen the wide application of Artificial Intelligence in these tools.

Organizations & Education

Despite the fact that departments and local entities have been established in different countries that take over the investigation of cybercrimes, one can still name internationally-recognized organizations. Their main goal is to offer professional training in the field of digital forensics, support investigations, and consult law enforcement agencies. In addition, there are other known resources and entities devoted to the education of digital forensics practitioners.

- *SWGDE* (Scientific Working Group on Digital Evidence)[2] was founded in 1998 and ensembles expertise from academia, law enforcement, and private organizations. SWGDE offers a discussion forum, issues best practices and guidelines, though does not perform any official certification in the field. They hold annual meetings.

- *NW3C* (National White Collar Crime Center)[3] is a non-profit organization that offers education in the field of digital forensics. Most of their courses touch a variety of aspects of Cyber Investigations and economic crimes.

- *IACIS* (International Association of Computer Investigative Specialists)[4] is another organization devoted to the training and certification of specialization in computer forensics beginning in 1990. They offer also several types of certifications.

- *SANS Institute* (Escal Institute of Advanced Technologies)[5] is a private company located in the USA that has taught information security courses since 1989. They offer a variety of hands-on practice in the areas of memory, mac, and smartphone forensics. In addition to this, they support the community and publish various materials that can be of help in Cyber Crime Investigations[6]. As of Spring 2017, there are eight 6-days courses in Digital Forensics offered by this company[7]

- *ENISA* (European Union Agency for Network and Information Security)[8] is an organization in the European Union devoted to network and information security training located in Greece beginning in 2005.

[2] https://www.swgde.org/
[3] https://www.nw3c.org/
[4] http://www.iacis.com/
[5] https://www.sans.org/
[6] https://digital-forensics.sans.org/
[7] https://www.sans.org/courses/forensics
[8] https://www.enisa.europa.eu/

- *DFRWS* (Digital Forensic Research Workshop) [9] was started in 2001 in order to create a network for law enforcement agencies, academic professionals, and investigators with the goal of discussing modern challenges and possible. DFRWS happens annually in the U.S. and Europe.

- *NIST* (National Institute of Standards and Technology) [10] is an agency in the U.S. that handles the standardization of various aspects of technology and science. Particularly worth mentioning is "Digital Forensics at NIST" by Lyle et al. [258] that gives an overview of the projects devoted to Digital Forensics.

- European Anti-Fraud Office[11] offers practical support and guidelines on Digital Forensics Procedures in European Union (EU). In addition to this there have been established a number of education programs related to Computer Forensics in EU[12].

Current Best Practices

In addition to variety of trainings performed by the companies and organizations mentioned above, we can also find specific certifications that are currently being offered. GIAC (Global Information Assurance Certification) was found by SANS Institute in 1999 and offers the following certifications in a field of forensics[13]:

- **GCFE** Certified Forensic Examiner

- **GCFA** Certified Forensic Analyst

- **GREM** Certified Reverse Engineering Malware

- **GNFA** Certified Network Forensic Analyst

Furthermore, there are certifications by IACIS:

- **CFCE** Certified Forensic Computer Examiner, which was one of the first training programs, initially introduced in 1998

[9] https://www.dfrws.org/
[10] https://www.nist.gov/
[11] https://ec.europa.eu/anti-fraud/investigations/digital-forensics_en
[12] http://www.forensicfocus.com/computer-forensics-education-europe
[13] http://www.giac.org/certifications/categories

- **CAWFE** Certified Advanced Windows Forensic Examiner
- **ICMDE** Certified Mobile Device Examiner

On the other hand, we can see a number of *guidelines, best practices and recommendations*. These are created based on previous experience, developed technologies, and need of the industry. Garfinkel [159] in 2010 sketches the possible needs and directions that Digital Forensics will require in the next 10 years.

- *Best Practices for Computer Forensics* by SWGDE [406] (originally from 2005) describes how the equipment preparation, acquisition, and examination should be performed. The documents contain recommendations of how the Digital Forensics Process can be properly documented.
- *Forensic Examination of Digital Evidence: A Guide for Law Enforcement* by NIST and U.S. Department of Justice [63] covers the procedures of policy development, evidence assessment, acquisition, examination, and reporting. The documents contain a number of explanations and examples of case reports.
- *A Ten Step Process for Forensic Readiness* by Rowlingson [341] from 2004 defines ten pre-emptive steps an organization may take to be able to handle cyber incidents in a timely manner. It also describes how preparation for digital evidence collection must be performed.
- *ISO/IEC 27037:2012* [14] describes different aspects of using digital data for computer forensics evidence examination.

Benchmark Datasets

Education in Digital Forensics plays an important role in the quality and timeline of an investigation. Therefore, besides the above-mentioned courses, there also exist datasets used for training specialists and testing relevant tools. One of these resources is *Digital Corpora*[15], which includes following data:

- *Cell Phone Dumps* - different APK files and other examples of malicious software that can be installed on mobile devices.
- *Disk Images* - a collection of images ranging from 2008 to 2014, representing different devices such as iPod and file systems like NTFS.

[14] http://www.iso27001security.com/html/27037.html
[15] http://digitalcorpora.org/

- *Files* - "Govdocs1", which is about 1 million different files sorted into 1,000 directories with 1,000 in each. The main idea is to provide the basis for the forensic analysis of files, including carving and file type detection.

- *Packet Dumps* contains information about DARPA Intrusion detection datasets 1998-2000, which also served as a basis for now obsolete KDD Cup 1999 data contest.

- *Scenarios* - a collection of Encase E01 and Raw scenarios that include different types of crimes to be practiced with.

As for malware analysis research, the following are two publicly available datasets:

1. VX HEAVEN [20] is dedicated to distributing information about computer viruses and contains 271,092 sorted samples dating back from 1999. The taxonomy of the collection includes categories like Trojan and Backdoors, and also divided by targeted operating system.

2. VIRUS SHARE [17] represents a sharing resource that offers 28,281,360 malware samples mostly unsorted as of Spring 2017. The first archive appears to be from 2012.

Types & Availability of Handled Data

We describe here another aspect related to data analysis in Forensics Science: the types of data used in different areas of Information Security. Coming into era of Big Data places challenges in front of the analyst in the form of multiple data structures that need to be analysed. Static data seized from the crime scene are no longer the only type of data involved in crimes. Dynamic behaviour data collection is a new trend in digital forensics applications such as network traffic analysis. This means that new knowledge paradigms have been introduced into all fields of Forensics Science when dealing with data analytics. Therefore, the following data type's classification is performed. SC methods will be specified in the next section according to this classification. Another important reason for this specification is the unique applicability of each method, since not all of them perform well in the same data type. According to the studied literature, we can distinguish the following most common data paradigms:

- *Static data / Constant availability* represents data that do not change their quantitative and qualitative properties during analysis. There is usually no time limitation and CI can use it multiple times.

- *Dynamic systems / Limited availability* takes place when it comes to the investigation of a crime scene in a dynamic environment or ongoing crimes. This paradigm considers a live system that has information and where different systems state change non-deterministically under the influence of some external factors. Moreover, such systems aggregate both data streams and static content. In Digital Forensics, Order of Volatility is applied to preserve as much valuable information as possible.

- *Data streams / Very short availability* implies an information "pipe" where the data changes as a nondeterministic flow and there is limited storage space to keep the old data. Data streams can be retrieved from various sensors.

Since the data available are raw, feature extraction and selection must be performed in order to reduce the dimensionality of the analytical models. Additionally, there might be a need to map raw values into numerical or more appropriate forms for model constructions. Methods of data pre-processing have to be fast and reliable. In the case of data streams, the data characteristics will change over time. Further, 5Vs of Big Data suggests that the velocity the size can be mitigated by the hardware and software solutions, while incomplete and unstructured information are to be handled by the CI methods only. As was studied by Quick et al. [329], the volume of case data in recent years has grown extremely, meaning that the human expert has a physical limitation when analysing it manually.

2.2.2 Automation in Investigation[16]

Cyber or Digital crimes is one of the recently emerging areas in Crime Investigations. It's emergence caused by the fact that the number of ICT-related devices is growing exponentially each year with the corresponding growth of the information being stored, processed and transferred [197]. In some cases, this leads only to passive observation of the activities, like malicious software execution, network traffic monitoring, and logs analysis as depicted in the Figure 2.1. This makes the process of investigation even more cumbersome and time consuming. Therefore, there is a need for advanced data analytics based on the compromised indicators and previously collected historical data.

Digital Forensics process

We refer to the master thesis by Puzyriov et al. from 2013 [325], where the comprehensive study of the investigation tools for DF were given. Basically, it shows the investigation phases used in Digital Forensics that can be mapped to other Forensics Sciences. As the author studied, the most common eight phases used in

[16] Ideas of this subsection are published under the contribution [367]

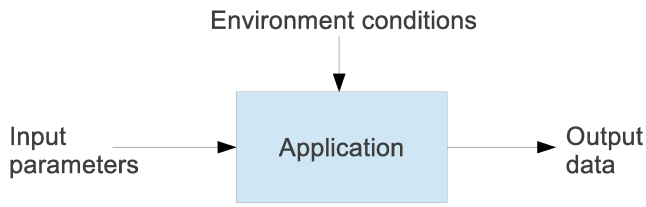


Figure 2.1: A typical way of black box testing for software analysis

DF Investigation process are: *Identification, Preparation, Approach Strategy, Preservation, Collection, Examination, Analysis* and *Presentation* [63]. The first four steps may vary from case to case when considering Forensics Science in general:

- *Identification* - define the goal of the ongoing investigation, possible suspects and victims.
- *Preparation* - prepare digital carriers to be seized according to chain of custody.
- *Approach Strategy* - specify the instruments and tools to be used in which manner and how.
- *Preservation* - preserve digital evidence without its alteration.

Cases may touch DNA samples, fingerprints, economic crimes, etc. Therefore we do not consider the aspects related to physical presense at a crime scene, since it is out of the scope of this thesis. Furthermore, we only concentrate on the last four phases that are considered to be data analytic-related. At this point, SC is considered as a data-driven approach to facilitate the investigation process as defined by Franke in the PhD thesis related to signatures verification from 2005 [150].

- *Collection* implies standard-based collection of evidence and relevant data.
- *Examination* targets the identification of possible evidence of interest.
- *Analysis* the most comprehensive and important step in the Investigation.
- *Data presentation* provides a way of describing found traces in a Court of Law.

The general perspective of the Digital Forensics process and peculiarities of each step are given in the Figure 2.2.

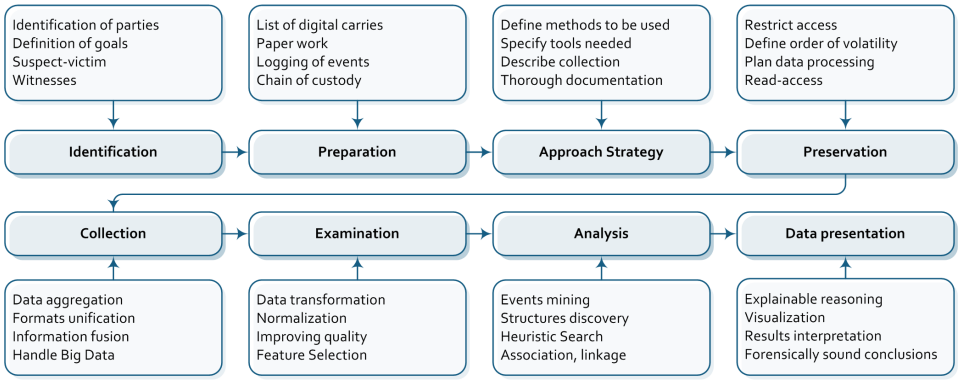


Figure 2.2: Details of different phases in Digital Forensics Process

Our idea in this work is to utilize the potential of Soft Computing to solve computer-related crimes. In the Figure 2.3, we have highlighted the fields in each of the phases that will benefit from SC application. On can see that SC applicability is limited mostly to phases where there is a need to analyse data, as it incorporates a family of data-driven approaches.

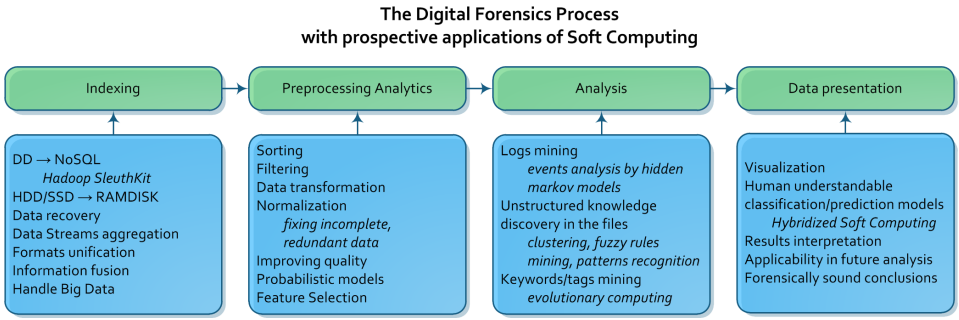


Figure 2.3: Possible application of Soft Computing for Digital Forensics

Computers in Cyber Crime

Computers have been heavily used over the last decade with the emergence of the Internet and Software as a Service (SaaS). One can create extensive list of the way malicious actions and attacks can be performed [296]. Generally speaking, there are two types of crimes involving ICT systems, computers in particular, according to classification done by Gordon in 2006 [166]. These crimes have completely different approaches and require a variety of skills to investigate and mitigate them.

Type I involved computer systems as a tool in Crime such as when stealing sens-

itive data by tricking a user, placing malicious software on the user's machine to perform spying or data logging activities, etc. Such crimes have mostly financial or political interest while instrumenting the attack.

Type II differs from the previous type by the role of a computer system as being the sole target in an attack. At this point, an attacker or any third party can use social engineering or exploitation techniques to get access to protected information or functionality. In this type of cybercrime, goals may angle towards getting profit as well as performing Type I crimes.

Computational Forensics (CF), also called Computational Criminology is a subfield of Forensic Science that uses various scientific methods such as modelling, analysis, pattern recognition, and data mining to facilitate the Cyber Crime Investigation with the help of methodology that can deliver quantitative results in a Court of Law. As detailed in the book [151], a number of files that benefit from CF is given, and includes Printer identification, human identification, shoe prints analysis, speech recognition and handwriting. Authors of the book analysed the contemporary state of the art in the field and concluded that artificial intelligence methods can facilitate Investigations in three ways: (i) better analysis of found evidence, (ii) improved processing of large-scale data, and (iii) representation of expert knowledge to be used instead of automated analysis. As a result, we see that CF greatly benefits Crime Investigation by means of increasing awareness, speeding up data analysis, and improving understandability of the data.

Toolset

A challenge with digital forensics is also with the growing number of file formats that need to be carved from the disk properly, taking in mind their headers and footers. Therefore, one can see an increase in a number of tool used for Digital Forensics over the last few years. We can notice that the main functionality includes a string of full-text searches in the image, file carving, and extraction of erased data. A Master Thesis by Puzyriov [325] describes a way of testing these tools. The author identified a number of stages in testing methodology required to verify the tools. Finally, a large number of practical tests were performed using different tools listed below, as well as a number of datasets reflecting real case scenarios. Another aspect of the Cyber Investigation is the so-called *Order of volatility* [147] that determines the volatility of the data in the computer systems such as disk files, registers, RAM, etc. The evidence should be collected according to this order using corresponding tools in order to avoid the loss of any relevant data independent from location and storage properties.

- *EnCase*[17] is a Windows OS-oriented software for digital forensics investigations focusing particularly on analysis and reporting.

- *FTK* (Forensic Toolkit)[18] is forensic software offered by AccessData that is designed to look through the data on HDD for traces of evidence.

- *The SleuthKit/Autopsy* (TSK) is a Windows/Linux software that assembles a variety of command line tools that can facilitate the extraction of metadata, timestamps normalization, etc.

- *SIFT* (SANS Investigative Forensics Toolkit)[19] is a Ubuntu-based toolkit that comes as a virtual machine image and can be deployed in a timely manner with already-installed software.

2.2.3 Challenges & Limitations

Despite the advantages and variety of methods for different tasks, we can see the general limitations of stand-alone methods that make their application challenging.

When dealing with digital forensics case data, it all usually comes down to the analysis of data collected on the crime scene in a forensically-sound manner. Since forensics science is a mostly data-driven area of methods, the investigator decides what methodology should be used and how. There may not be any useful information found since manual analysis is limited and can't overcome the limitations circumvented by an automated one provided by CI. There have been several studies conducted by Mitra et al. [285] in 2002 and Karray et al, [216] in 2004 on the SC models applications involving different types of data used. Though comprehensive studies, it does not provide an overall view on the feasibility survey of SC in Forensics Sciences. Additionally, it can be stated that there are difficulties in the construction of automated systems such as decision support engines. This is because of the inexact solutions that SC provides. Franke et al. [150] presented a view on SC as the most suitable application respective to the increased Big Data demand and trade-off between required accuracy. Despite crisp decisions, it provides a rather fast and rough answer involving degree of likelihood of the event or probability of the actions undertaken. Another challenge is its inability to handle large-scale and multinomial data analysis problems that would probably result in rather erroneous results, and is not suitable for application in DF in standard form.

[17] www.guidancesoftware.com

[18] http://accessdata.com/solutions/digital-forensics/forensic-toolkit-ftk?/solutions/digital-forensics/ftk

[19] SANSInvestigativeForensicsToolkit

To mitigate this, one of the commonly used solutions that can be found in the literature is the combination of several stand-alone ML methods with the hope of mitigating existing weaknesses and enforcing strength and resilience against errors. This is work towards so-called Hybrid Intelligence (HI). Kamar stated that the human intellect becomes an inseparable part of such systems [212]. Abraham gave an overview of the HI employing a fuzzy inference system [40]. Inspired by this work, we believe that application of HI for Digital Forensics can bring benefits related to faster data processing and improved understandability of the decision derived by Machine Learning.

In this subsection, we describe the challenges with the usage of ML for Digital Crime Investigations.

Human Rights & Privacy Preserving

Human rights denote a set of principles and rights that belong to each human. When dealing with Crimes Investigation however, several issues and concerns may arise, especially when applying automated data processing. Saleem [348] investigated the protection of evidence and human rights during the Digital Forensics Investigation. The authors argued the importance of preserving basic human rights in the trial guided by a need to guarantee proper evidence handling and following the DF process. One of the key ideas is the need to preserve the integrity of evidence found on a crime scene. Another work by Saleem et al. [349] described the current state of the art and concluded that some of the published frameworks lack proper digital evidence protection with respect to human rights preservation. Another aspect that one should be concerned with is whether automated data processing can infringe upon human rights. This comes from the fact that intelligence methods can be trained using biased historical information or the expert's knowledge. Chen [94] discussed the importance of Computer Forensics and a need for law to be able incorporate technological advances with respect to human rights principles.

Privacy is one fundamental human right. Many countries created privacy acts, which are intended to guard data and sensitive information from being disclosed or unlawfully used by other parties. Norwegian Personal Data Act [21] describes the principles of data processing that should also comply with the basic human right for privacy. The challenge with Digital Forensics comes when the investigation is performed over the seized data, which may contain some private information irrelevant to the case. Best practices define so-called access policies for forensics data: *private* and *non-private* [174, 242]. A person involved shall put flags on the data that are considered his sensitive private information, so the Forensics Investigator has no right to look into such data. However, aforementioned categories

may be divided into *relevant* and *non-relevant* data meaning that it is user's choice to whether the data has to be collected if it is relevant and private with respect to his right for privacy. This may be hard to deal with once an investigator employs automated data processing methods such as Machine Learning. On the other hand, this creates a way for cyber criminals to use variety of data encryption solutions, resulting in the crime investigation failing to retrieve any data at all [72, 445].

Ethical Issues with Machine Learning in Digital Forensics

Another concern has to do with the extent Machine Learning can be used for decision making in Digital Forensics Investigations along with human expert skills. It might be infeasible to perform manual analysis by a human expert, even taking into account the human resources of considerably big investigation departments and law enforcement agencies [1]. We can see that ML can be used as a decision support mechanism. Right now there are debates about the usage of automated analysis of the data in Digital Forensics [204]. Because of such difficulties with the complexity of the Big Data, the utilization of human resources becomes less efficient. However, evidence should still be presented to the Court of Law so as to clear one or another side. This means that if the data analysis cannot be performed by human resources, the computer data processing power and ML should take place to do the routine job in a faster manner. As an alternative method for *Analysis* and *Representation* of the Digital Forensics Process, we consider AI, which is a set of methods for analysis and learning from data in order to make a decision or represent the data in human-understandable way.

Advantages of ML usage in Digital Forensics

1. *ML can help humanity by processing large-scale data.* There are may prospective studies that show that AI is a powerful tool that can help mankind simplify life. One fascinating example is the British theoretical physicist Stephen Hawking whose voice is completely generated with the support of an AI engine developed by Google [28]. So, as AI helps in such complicated problems related to everyday life, it can be suitable for a narrow data analytic as well. This statement is supported by a new technology based on AI developed at Facebook AI Research lab[20] that will help users in Facebook guard their privacy and sensitive information [317]. AI seems to be useful from a moral and ethical perspective in maintaining mankind's wealth, and can be ethically used for Digital Forensics. Finally, Hallevy adds to this argument through the extensive study in his book "Liability for Crimes Involving Artificial Intelligence Systems" [176], in which he argues that AI

[20] https://research.facebook.com/ai

can be plausible in imposing the liability of the AI methods in trial process. Yet the main point was that the legislation system has to comply with modern technologies along with the secular statutes. Thus, the human cannot be replaced by the machine or AI to make a decision regarding law-related issues. At this point not all data analysis tasks in Digital Forensics may be solved by human expert. Therefore, it is simply necessary to apply advanced automated techniques in order to extract some of the meaningful evidence from Big Data in order to be presented in a Court of Law.

2. *Human experts can do more important jobs than solving complex data analytic problems that can be automated.* The challenging situation arises when it comes to the decision whether the analytic work should be done manually with a higher degree of accuracy, or automatically with higher processing speed, though even a hard working analyst can miss important clues that lead to vital evidence. According to author [100] working hard and working ethically are quite different things. Following this we can state that human workers can misuse work and do unethical things like claiming to have performed more work than was actually done. At this point, AI works as it was programmed, and provides results which can avoid ethical problems since it is usually a deterministic automaton.

   ML can be used as a methodological approach if they are accepted and follow Daubert Standards. Daubert Standards describe testing models and interpreting them according to the stated requirements [146]. As law is established in a set of commonly accepted and discussed rules, Cyber Crime Investigations are based on these rules, and corresponding investigation methods are developed. Thus, one can refer to the legal precedent in the USA in 1993 that caused the emergence of the so-called Daubert Standards. These standards define whether the testimony and derived evidence can be accepted in a Court of Law as valid. The first criterion defines whether a method is based on a testable hypothesis. It can be said that ML methods such as classification are based on the hypothesis, while those based on the data automatically make a decision whether to accept or reject a null-hypothesis. The second criterion outlines that the error rates should be known for purposes of the scientific method. The third criterion requires the ML method to be peer-reviewed. As ML is actively developing and testing new methods and applications, there are plenty of publications on ML methods with results for different areas that can be referenced. Finally, the fourth criterion states that the scientific methods have to be accepted in the community. The main problem lies in the need to present data analysis in a "forensically-sound manner" as mentioned by [275] without changing the original data

properties (changing evidence). Most of the existing methods generate a large number of rules that do not comply with Daubert Standards and do not provide a reliable human-understandable model. When dealing with case data, it usually comes down to the analysis of data collected on the crime scene in a forensically-sound manner. Forensics science is mainly concerned with data-driven methods, so investigators must decide what methodology should be used and how. Since a manual analysis is limited and in many cases cannot compete with an automated one provided by ML, one might not be able to find any useful information through its use.

Considering arguments for use in a cybercrime investigation, we can say that the numbers 2 and 3 are the strongest points for use of AI methods. Other points can be eliminated since they do not influence ethical decisions in bringing results and making decisions affecting human destiny.

Disadvantages of ML application in Digital Forensics

There are several strong objections to using the ML methods. Despite the fact that such applications may bring many benefits and are scientifically accepted, we can name multiple issues related to AI that put a big question mark on its place in the investigation process.

1. *ML methods may fail to outperform human.* Beginning at the end of 1960th the enthusiasm for application of AI in different spheres of a human life waned. That period denoted a so-called "AI Winter" that has lasted until now [23]. Persistent failures questioned the new technologies and reviewed them under new and more aggressive assumptions, revealing that AI may work under some very specific and sometimes unrealistic constraints.

2. *Legislation-related positions cannot be automated and require human expert.* The first major considerations regarding the place of AI in human life were laid by Joseph Weizenbaum in his book "Computer Power and Human Reason: From Judgment To Calculation" [435]. In his work, he distinguished two main activities in decision making process, which are inherent to both humans and AI: deciding and choosing. Deciding mainly includes AI activity that is based on some previous information and prior factors. Choosing can be performed by human and also implies the involvement of some moral factors rather than the use of purely historical data. Weizenbaum stated that above all, the human-occupied positions of judge and police officer cannot be replaced by a computer intellect. There are however at least two successful applications of ML in Crime Investigations. The branch of AI, Soft Computing, was used in face recognition based on

the fuzzy position of people's heads in photos [109]. Another one is a tool called COPLINK[21] that employs AI to help police officers to find suspects in a particular crime in a particular geographic areas.

3. *ML methods result in hardly-explainable solutions.* The results may be poor and hardly explainable as it happens with Artificial Neural Network or Genetic Algorithms. In these methods, nature-inspired and human brain-alike analogies were used. The results from such methods however are almost impossible to interpret by humans and give in an understandable format. If it is so, it can cast doubt the AI decisions to be moral with respect to the direction of human destiny. In order to overcome this challenge, the Hybrid Intelligence (HI) takes places making advanced AI methods more understandable. For example, Artificial Neural Networks (ANN) can be combined with Fuzzy Logic to creates Neuro-Fuzzy methods. Such methods provide simple human-understandable rules to describe the decisions made by ANN. Thus, the above-mentioned objection can be easily mitigated by HI.

Summary. To summarize, the use of ML methods have no moral issues in cybercrime investigation make the world a better and safer place, yet it should not take over control of important aspects of everyday life from humans. We can say that the Computational Forensics can be used as a valid set of scientific methods that comply with Daubert Standards and provide reliable, human-understandable results. Thus, the hypothesis defined earlier is accepted. The complex data analysis problem can be solved with the help of AI methods. It should be clearly stated however that one can consider moral the actions done by AI as long as they do not perform self-evolving movements uncontrolled by humans. This is a crucial factor, since right now humans possess higher intelligence over others that allows for ruling the world and controlling the global process. The AI cannot decide for people, but it is ethically good to support a decision of the criminal investigator to speed up the process and reveal as much evidence as possible on the seized data carriers. Additionally, when the methods are approved by communities and well-tested, they can be easily used without any additional inventions as long as they perform deterministic actions on the presented data in a human-understandable way.

2.3 Machine Learning & Advanced Analytics[22]

Machine Learning (ML) is a sub-field of Computer Science and describes the ability of a machine to learn from previously collected data (experience) E over the

[21] http://www-03.ibm.com/software/products/en/coplink

[22] The main ideas of this section are published under the contribution [367]

same defined set of problems T and the performance attribute P with the inevitability that over the time P is improves. This is also called a well-posed problem according to Mitchell [284]. There exist several fundamental tasks that CI can solve as stated by Fayyad in 1995 [144]. Each specific method is applicable for a particular task, which means that the corresponding guidelines on correct application can be built upon. According to this, we can state that CI solves the following problems, including their overlapping based on historical information or information derived from a case:

- *Classification* arises when the data has to be separated into groups based on their properties.
- *Clustering* aggregates data samples according to the similarity in properties without any prior information about the groups.
- *Regression / Forecasting* makes a statement about a future or unknown event considering available historical information.
- *Rules learning / associated rules* extracts common characteristics of groups of data samples and represents them in a general form suitable for other needs.
- *Optimization* gives an opportunity to look for an optimal solution based on different criteria and defined constraints. GA and SI are considered to be meta-heuristic optimization methods where the information is inaccurate or incomplete.

Generally speaking, all methods use the following routine depicted in the Figure 2.4 also describe by Kononenko et al. [232]. At the same time, we have to differentiate between general *Statistics* and *Machine Learning*. *Statistics* is intended to help represent and interpret given data in order to understand the properties for a proper selection of the *Machine Learning method* [85]. At the same time, this process can also be called *data modelling* using a specific function and corresponding tests to validate the correctness of the model [153]. Contrary to such modelling, *Machine Learning* offers "algorithmic modelling", where the model is known, yet during the learning process one seeks to find the best function that fits the data according to the defined algorithm. When it comes to real world data analytics however, we have to rely on preliminary data analysis using Statistical methods for a proper selection of the ML methods and corresponding hyper-parameters.

Depending on the task to be performed, we can separate between two main stages:

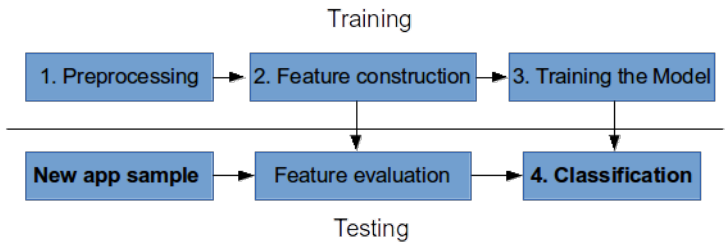


Figure 2.4: Dataflow in a general Machine Learning approach

Training - a process of fitting the model to data under specified performance measures, showing ability of the model to represent the given raw data.

- *Data preprocessing* - a process of transformation of raw data/characteristics (like network traffic, software analysis, log) into a format suitable for further processing.
- *Feature construction* - a set of methods to construct features and select the most relevant and non-redundant features
- *Model selection and training* - ML algorithm has to be selected with specific parameters tuned and ready for training according to the defined ML task.

Testing - a process of model evaluation against new samples to find out a particular group it belongs to or forecast unknown parameters.

- *New data arrival* - a new raw data that needs to be converted using routing defined in the Training's first stage.
- *Features evaluation* – an evaluation of the new data against a set of crafted features.
- *Classification / Regression* - a model that takes previously evaluated features

In reality however, the mentioned above scheme comes to a more advanced dataflow that requires (i) relevant expertise and (ii) data to be able to generate a model capable of delivering the required performance on given historical data. Therefore, it is important to understand that the correct application of ML to solve data analysis problems requires proper utilization of each particular method as shown in the Figure 2.5. This is the key factor when we consider not only the performance but also the trade-off between accuracy, complexity, and computational time.

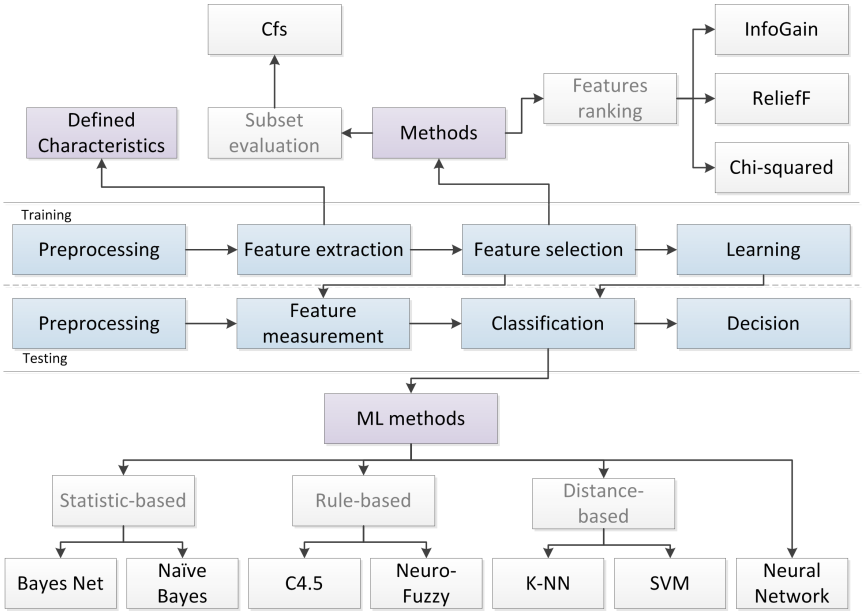


Figure 2.5: A overview of the possible methods to be used in a general ML approach

2.3.1 Hard & Soft Computing

Starting from the 1990's in the literature, we can see a specific separation between the methods into newly formulated *Soft Computing* and conventional *Hard Computing* according to work by Zadeh [455]. In this work, the author presented a paradigm of human-alike computing capable of handling mistaken and missing data by generating models with inexact solutions. Zadeh advocated that Soft Computing may help people to solve data analysis problems related to real-life, where it is important to produce human-understandable reasoning of the decision making process given by ML methods. Below, we present the weaknesses and strengths of both paradigms considering all ML methods [7, 232].

Hard Computing is a conventional paradigm, where exact values of the parameters are used to produce a model describing the data. It is also called precise learning or *classical artificial intelligence* [304]. *Advantages:* produces high accuracy data models; precise answers to defined questions; deterministic; operates with binary logic (TRUE or FALSE) only. *Disadvantages:* requires manual tuning of the parameters of the model; mostly sequential learning; cannot be used for real-world raw data analysis; large computational requirements. *Methods:* C4.5, Random Forest, Random Tree.

Soft Computing introduces approximation by utilization of inexact solutions in

data modelling. This set of methods can also be named as *computational intelligence* since here we are dealing with tolerance and a trade-off between model understandability, generalization and imprecision. The foundations of the SC and Fuzzy Logic in particular were studied in depth by Zadeh starting from 1960th. From the literature review we can say that there are several SC methods that already found an application in Forensics Science. The following groups of methods were selected. PROBABILISTIC MODELLING (PM) is a set of methods that are capable of solving multiple tasks, including explanatory data analysis, especially when dealing with uncertainty [310]. Moreover, PM provides a flexible support in an understandable way for decision making systems. Next important group is APPROXIMATION, which covers methods capable of learning a model from data that is an approximation of given properties [26]. Moreover, SC can not only offer reasoning and inference, but also optimization, in particular METAHEURISTIC. It gives a wide range of methods that can provide mathematical optimization. We can see that SC contains a variety of methods that can automate evidence processing and analysis. *Advantages:* handles real-world data with mistaken or erroneous attributes; accepts partial truth (whole interval [0;1]); low computational cost in addition to ability to apply parallel optimization; linguistic human-understandable answers can be derived. *Disadvantages:* can produce lower precision models while dealing with data interpolation and mistaken or noisy entries; may require precise parameters tuning involving human expert. *Methods:* Naive Bayes, Bayesian Network, SVM, ANN, FL.

Publicly Available Tools

Today, machine learning is widely used in many areas of research, and the use of already made tools (Software products, libraries etc.) is an important part of ML usage.

Weka or Waikato Environment for Knowledge Analysis is a popular, free, cross platform and open source tool for machine learning. It supports many popular ML methods with possibility of fine tuning of the parameters and final results analysis. It provides many features such as dataset split and graphical representation of the results. As an output, it uses .arff file format which is specially prepared CSV file with header. It has several shortcomings: no multi thread computations, and problems with big dataset because of badly optimized memory usage.

Python Weka wrapper is the package which allows the use of Weka from the Python programs. It uses javabridge to connect Java-based Weka libraries with python. It provides the same functionality as Weka, but allows the better automation of the research process. The author maintains the community where everybody can get fast and useful help.

LIBSVM and LIBLINEAR are open source ML libraries written in C++ and that implement kernelized support vector machines for classification and regression, and linear SVM. Bindings for Java, Matlab and R are also present. It uses space-separated files as input, where zero values need not to be mentioned.

RapidMiner is a machine learning and data mining tool that is present in free and paid versions. It provides usable GUI with support of a lot of ML and data mining algorithms.

Dlib is a free and cross-platform C++ toolkit with support of many machine learning algorithms. It supports multi-threading and has Python API.

2.3.2 Decision Support using Binary & Fuzzy Logic

The construction of rules has been an inevitable part of the decision support process, where expert knowledge can be transformed into mathematical terms [337]. There are several approaches for how the decision rules are built depending on available data and required form of the decision. Below, we give an insight into the origins of Binary Logic and Fuzzy Logic and why it is important to have a concept of *partial truth*. For the simplicity of the concept explanation, let's consider an example where we need to infer the *Feeling* f_1 measure of whether to be outside on the street or not, based on the *Temperature* t_1 as one of the variables.

Binary Logic is classical logic from Boolean algebra that operates with truth measures that are either completely FALSE (0) or completely TRUE (1) [297][Chapter 3]. This type of logic is also used in Computer Systems. It is likened to a simple switch that can have two states: either OFF (0) or ON (1). Such logic is simply reproducible by any trigger elements based on vacuum lamps or transistors. To create the example of the rule is given below:

$$IF \;\; t_1 = 10C \;\; THEN \;\; f_1 = Cold \tag{2.1}$$

Where t_1 is a numerical independent variable that is explicitly compared with a the exact numerical value of the temperature. Furthermore, if the statement in the antecedent part IF of the rule is True, then the consequent part $THEN$ defines value of dependent variable f_1. One can see that in case of multiple variables and the presence of specific intervals in the independent variable, the construction of a large amount of equations may be required to be able to make a decision properly.

MODEL STRENGTH: avoids ambiguity and offers low computational complexity.

MODEL DISADVANTAGE: incapable of handling real-world data since the resulting model is too complex to be utilized.

Fuzzy Logic is a concept of logic building under the assumption that the truth can

have a partial value. In contrary to classical Boolean logic, Fuzzy Logic operates with the whole interval of truth values $[0; 1]$ instead of binary FALSE (0) and TRUE (1). This is possible because of the concept of *degree of truth* or *partial truth* that was described and introduced byZadeh [455]. The general form of the Fuzzy Rules is as following:

$$IF \;\; t_1 \in LOW \;\; THEN \;\; f_1 \in Cold \tag{2.2}$$

One can see a similarity to the conventional decision rules, while the difference is that the numerical input variables are checked against specific terms of fuzzy sets rather than single crisp numerical values. So, basically *a numerical variable* is transformed into *a linguistic variable*. This linguistic variable represents a *fuzzy set* with a finite defined number of *linguistic terms*. Each of the terms reflects a specific range of numeric values from the input variable. The overall process of fuzzy rules application by multiple steps is shown in the Figure 2.6, while the steps are described below.

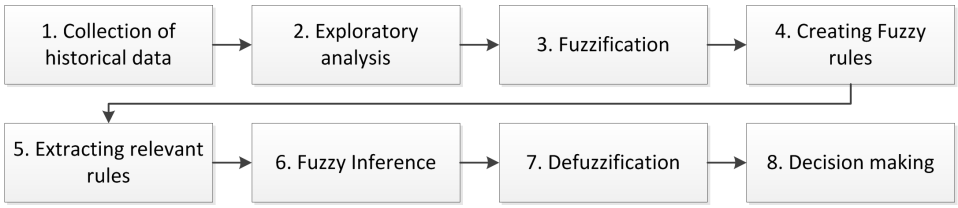


Figure 2.6: Fuzzy Logic process

1. *Data collection* includes the process of the collection of data characterizing a particular object or event to be able to extract mean values according to the central Limit Theorem [434].

2. *Exploratory analysis* covers the discovery of statistical properties of the target variable based on the historical information to establish a baseline.

3. *Fuzzification* is the transformation of a crisp numerical variable into corresponding linguistic terms of an arbitrary fuzzy set as shown in the Figure 2.7.

4. *Fuzzy rules creation* is the process of locating multi-variable dependencies in the fuzzified data in order to establish an IF-THEN relationship expressed in linguistic terms.

5. *Rules selection* defines the process of eliminating irrelevant and redundant fuzzy rules to reduce the complexity of the model.

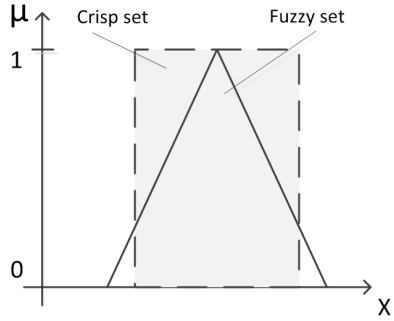


Figure 2.7: Comparison of crisp and fuzzy sets

6. *Fuzzy inference* describes a general process of transition from input crisp numerical parameters to fuzzy output variables such that class values of other linguistic attributes. This is done via evaluation of the rules in parallel.

7. *Defuzzification* is a specific process of mapping from linguistic terms back to crisp numerical attributes in the output layer.

8. *Decision making* is a controlled approach of the utilization of fuzzy rules for taking one or another action based on the numerical input parameters.

MODEL STRENGTH: handles real-world data, human-understandable, with a high degree of generalization.

MODEL DISADVANTAGE: requires manual tuning with the parameters carefully chosen to avoid under- or overfitting.

2.4 Use Cases in Information Security & Forensics

Over the years, researchers have developed a number of applications of Soft Computing to solve and mitigate different problems not only in Digital Forensics, but also in Information Security in general. Below, we give general examples of the problems and possible stages where it is important to apply Computational Intelligence, Soft Computing in particular, in order to gain understandability and the ability to classify traces of evidence automatically. The general domains that can benefit are listed below.

2.4.1 Windows Malware Analysis[23]

Taking into consideration the history of Windows malware beginning with the 2000's, we can see that this is the period where most of the modern malware categories were discovered in addition to an almost exponential growth in the number of Windows malware. In the first place, the proliferation was a result of the popularity of Windows NT families and its universal application in both private and public sectors, and critical and entertainment infrastructures. According to the study [247], around 80-95% of ATMs in the world still run Windows XP 32bit, which can have a large number of exploitable vulnerabilities. To tackle this, Microsoft forces banks to upgrade to Windows 8, though this might not be a realistic option due to transitional costs and the need for future support. Despite some security fixes, the Windows NT family was under attack due to outdated installed software like Internet Explorer 6 and unpatched versions of the OS. Around 2005, there was a peak of malware developments for Windows NT family considering the usage of IE 6 & IE 7 explorers [9]. One of the main reasons for such a tendency lies in user negligence and unawareness about security problems in Windows, as well as novel approaches used by attackers to trick consciousness by any means. There are general purpose systems as well as mobile- and server-specific editions. We can see from Kaspersky Security Bulletin [102] published in 2014 that despite the new versions of Windows 7 and 8, many users still use Windows XP. Moreover, majority of such users use 32 bit versions of the OS. More specifically, Windows XP Pro was installed by 23.7% users and Windows 7 and 7 Home by 19.74% and 5.96% respectively. The usage of Windows 8 was still under 5%, which can be explained by overall usage of Windows XP not only by home users, but also in corporate environments and marketing as well as some critical infrastructure such as power plans and backing systems. The official support of the last one ended on April 8th, 2014, forcing users to upgrade to the next versions according to the official Microsoft page [30]. According to a recent study by Net Market Share in September 2015, the overall usage of Windows XP went down to 12.21%, while Windows 7 shares as many as 56.53% of installations [388]. The 64 bit (x86-64) architecture is out of scope because its relative novelty (XP and Vista first offered full support) and predominance among software written for them. Most malware still use x86 architecture in order to be compatible with older OS versions. According to the survey, the number of installations of Windows XP 32 bit in 2010 exceeded 99%, and Windows Vista 32 bit almost 89% [244]. Additionally, it is sometimes impossible to port old 32 bit libraries and executables to a 64 bit version due to lost source code or tuned functionality. Finally, MS-DOS executables and 16 bit binaries are out of our scope. All malware can be separated into different

[23]The ideas of this subsection are published in the contribution [382]

categories based on their functionality, targets, and goals.

Malware or *malicious software* is a piece of software whose intention is to bring harm to a computer system or its users. It includes unauthorized access and the alteration and misuse of the information. However, malware can belong to different categories based on their functionality, targets and goals. Generally speaking, we can name the following categories of malware based on their internal functionality [79, 136, 217, 316]:

- *Boot and file viruses* started appearing from the end of the 80's and caused much harm to Microsoft OS, including early versions of Windows 3.1, etc. The systems were not designed to be intrinsically secured, so this caused a raise in number of adversaries.
- *Macro viruses* targeted earlier versions of Microsoft Office products, in particular exploiting the usage of macro languages. Recent versions of MS Office such that 2003 are more protected against such malware.
- *Mail viruses and worms* became popular with wide development of the Internet in the early 2000's when attackers attached different types of files containing malicious payloads.
- *Network viruses and worms* gained popularity in the middle of the 2000's when adversaries exploited different vulnerabilities and weaknesses in network protocols to gain access to systems.
- *Spyware* is software that targets information collection about a particular person or organization in a covert way. It can include various aspects of everyday computer usage, etc. A common way of distribution is through old versions of Internet Explorer.
- *Rootkits and RATs* are more sophisticated malware angled toward gaining remote access to a computer. We can see that many Advanced Persistent Threats succeed due to infection by a Trojan that installed the RATs.
- *Scareware or more correctly extortionware* intends to scare a user in case he uses unregistered software and explicitly forces him to download potentially malicious program that is described as having benign functionality.
- *Ransomware* blocks the user's system and requires a particular payment to unblock, stating 'legitimate' reasons.
- *Data stealing Trojans* became recently popular since users tend to access multiple resources in a cloud via a browser, which means that most of them store their credentials in the settings.

There are many ways malware infects the computer, however the human factor plays a crucial role in this process [136] as presented in the Figure 2.8. An example of a general infection that one can get through the Internet is given below:

1. A user request believed-to-be-benign file from some server or web-site that looks legitimate.
2. Request is redirected to a malicious server. This can happen when the resource being requested is hacked or an attacker owns this resource.
3. File that is return to user contains malicious payload or is a malware itself and user installs it in the system, possibly granting high-level privileges.
4. Sensitive data are transferred or modified (depending on the attacker goal) and then sent from the infected computer or devices.
5. An attacker has reached their goal by accessing the data or performing some other malicious activity.

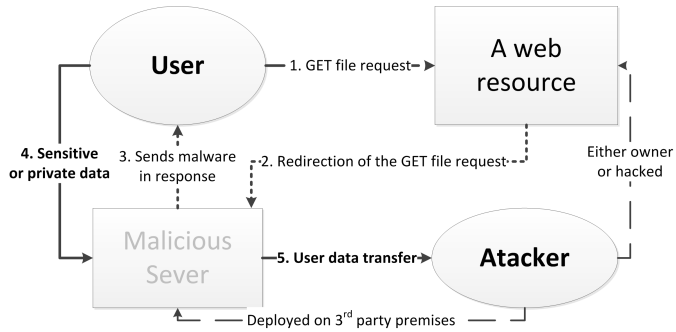


Figure 2.8: A general scheme of malware distribution on the Internet

Moreover, there exist a number of different platforms that can be infected, such as Windows (32 or 64 bits), Linux, OSX, Android, etc. As a result, the range of vulnerabilities and possible attack vectors is even larger when ICT system includes multiple devices with the aforementioned architectures.

To combat malware, there exist several commonly-accepted detection approaches including signature-based misuse and behavioural-based anomaly detection [196, 265]. These signatures can be MD5 hash sums of the files or more specific system artifacts, e.g. function calls, traces or specific stored information in the registry. Such signatures are very specific and are maintained mostly by malware analysts or reverse engineers. The antivirus software is hardly able to deal with

a dynamically changeable and proactive environment when using signature-based detection [110]. The main reason is that the signatures are composed manually, and therefore have little chance to deal with the possibilities of malware polymorphism and other obfuscation techniques. By using Machine Learning (ML) technologies over manual signature composition, generalization ability can be achieved when building detection rules. At the same time, we know that the Neuro-Fuzzy system has been used for defeating malware before [54]. So, we can see that malware detection and protection in the dynamic environment requires an intelligent approach to be able to mitigate risks related to data breaches.

Furthermore, a number of malware analysis techniques were developed in the years before 2010 according to Kendall et al. [221]. In this work, we target static properties analysis for malware executables. We believe that such methods have a lot of benefits when it comes to large-scale fast analysis of software samples using ML techniques. Additionally, there is a natural proclivity for attackers to stick with detection-evasion techniques that are usually used to prevent dynamic and static behavioural analysis, i.e. debugging, execution in virtualized sandboxes, etc. as studied by Marpaung et al. [265]. Moreover, static properties analysis does not require the installation of a specific OS or software, and is not influenced by the variety of software versions. Moreover, some of the tools for static analysis are already available in the OS, though there might be some limits to which static analysis still helps to differentiate goodware from malware according to Moser et al. [288]. Also, this type of analysis requires a lower amount of effort to build an environment, and eliminates a need for DMZ or other environmental precautions. From the literature review, we can see that many researchers investigated a variety of static characteristics, with the possibility of applying ML methods in mind. There is yet however no extensive evaluation of ML methods using the same datasets. In most cases, authors use small-scale, manually crafted collections of malware and goodware that are not scalable. Moreover, we can see that many works focus only on a single feature construction method and do not compare it to others. Thus, there is a lack of comparative studies of ML-based static malware detection. There are several ways of performing malware analysis depending on the types of characteristics and corresponding tests. We can highlight the following main approaches in malware analysis [221]:

- *static properties analysis* aims to study characteristics of malware files without executing them. Different aspects of files can be investigated, such as headers, possible encrypted parts, present strings, bytes, opcodes, and API n-grams, Portable Executable header features, strings, and others [167, 356, 422, 452].

- *dynamic behavioural analysis* considers different parts of the executed malware sample influence of difference factors present in the target system as shown by Kendall et al. [221]. Multiple activities such as network traffic, registry keys and disk usage patterns, API-calls and instruction tracing, and memory layout investigation are explored to find what differentiates malware from non-malware according to Egele et al. [134]. To collect such information, one can use either specialized sandboxes like Cuckoo [170] or utilize any Virtual Machines such as VirtualBox accompanied by monitoring software.

The big concern about executing malware by means of dynamic behavioural and static behavioural analysis in a safe environment is the possibility of evasion. These two analysis methods unfortunately are susceptible to evasion by malware that are aware of execution conditions and the computing environment. According to Lastline Labs [240] there are several common approaches used in malware. The most common is *environmental checks*, which malware performs before executing a malicious payload. These can be checks of virtualized environment indicators like QEMU, IP addresses, attached disk storage, etc. As a result, malware will intentionally prevent itself from executing in a set of sandboxes. Another approach is so-called *stalling code* that malware developers employ to delay the execution exploiting time constraints of automated malware analysis in virtualized environments. To simplify the data extraction routine from PE, multiple tools have been developed to assist the analysis. We consider only tools used to deal with PE file:

1. PEFRAME [57] is an open source tool specifically designed for static analysis of PE malware. It extracts various information from a PE header ranging from packers to anti debug and anti vm tricks.

2. PEFILE [87] is a multi-platform tool for Python used to discover various features, from PEiD signatures to detection packers, crypters, and compilers[12].

3. PE STUDIO [13] used to perform Malware Initial Assessment, delivering initial indicators in human-friendly format, virus detection using Virus Total, exploration of embedded items, etc.

4. IDA PRO [185] is a multi-processor disassembler and debugger used to generate assembly language code from machine executable code.

5. EXIFTOOL [179] is a multi-platform command line tool for reading metadata from different types of files, including exe and dll files, which are commonly used by malware developers.

6. HEXDUMP is a standard tool available through command line in Linux and is used to display a file in specific format like ASCII or one-byte octal.

7. OBJDUMP is a standard tool available through command line in Linux. It shows variety of information related to used instructions, addresses, etc.

Static analysis covers a range of techniques to dissect malware samples and to gather as much information as possible without executing the file. One of the commonly used tools is the *VirusTotal* [18] online scanner described by Seltzer [457], which provides output from different scanning tools, and also checks the submitted sample against 65 anti-virus databases and then gives the detection ratio as well as the names that each vendor has labelled the sample with. Another is the *PEframe* [57] that is able to detect packers, anti-debug, and anti-VM techniques as well as URLs and filenames used by the sample. *PEframe* and *strings* are somewhat redundant, the different tools can yield different information as, i.e. *PEframe* gives better structured output. Reverse engineering is a growing discipline that performs this dissection thoroughly in terms of using software to generate the assembly instructions and then be able to determine the actions that sample performs on a system as given in the Malware Cookbook by Ligh et al. [251]. We will not use this approach in our work however. Due to increasingly used and more complex obfuscation techniques, static malware analysis is becoming increasingly difficult to perform, as studied by Gavrilut et al. [160] and Idika et al. [196]. The biggest advantage of static analysis methods however are that they are considerably quicker, making it scalable and environment-independent.

Additionally, some services are provided online such as VIRUS TOTAL [18] that includes (1) analysis reports from different anti-virus vendors with detected malware names, (2) file and *exif* metadata, (3) comments on a particular file. However, the collected detection information might be subjective due to false positives.

After the preliminary literature study, we performed an extensive review of relevant scientific works. First, we concentrated on the malware categories applicable to Windows NT OS family and described existing malware analysis techniques. Then, an in-depth overview of static malware characteristics was given, including corresponding feature construction methods. Furthermore, we created a taxonomy based on the comparison of the used cases of ML methods, which will be given with guidelines for usage and utilization of available implementations. In the second part of the work, we gave a tutorial on how these methods are applied in real case large-scale scenarios based on the publicly available datasets for both malware and benign software collected from Windows OS (XP, 7, 8, 10). Later on, collected data will be pre-processed, static features will be extracted and ML methods will be applied. Finally, we discussed applicability of each particular method

based on the achieved accuracy. So, this work contributes as a survey of the existing state of the art in malware analysis using machine learning. We believe that static analysis has great potential and can facilitate large-scale malware detection.

It is difficult to determine exactly who wrote a computer virus due to various reasons, including the difficulty of tracing back to the developer. On the other hand, many laboratories and other enthusiasts extract and share recent malware samples on the Internet for community or research interests. We would like to mention the two following publicly available datasets for Windows malware that we are going to use in our research as per October 2015:

1. VX HEAVEN [20] is dedicated to distribution of information about the computer viruses, and contains 271,092 sorted samples dating back from 1999. The taxonomy of the collection includes categories like Trojan and Backdoors and is also divided according to their targeted operating systems.

2. VIRUS SHARE [17] is a sharing resource that offers 23,626,011 malware samples mostly unsorted. The first archive appears to be from 2012.

Additionally, we can mention tools that are used to collect recently appeared malware on the Internet based on daily updates from different sources. One of the tools that can be used to harvest malware as well as categorize them is MALTRIEVE [269]. The advantage of this tool is that there is a high chance of obtaining a malware that has been recently developed, yet the amount of malware and collection speed is slow compared to the earlier dumps referred to.

*Dynamic behavioura*l malware analysis can be done either by using Sandboxes or Virtual Machines accompanied by a debugger or other watchdog software. There is also a likelihood of revealing an internal functional logic. Contrary to this dynamic, static analysis may fail to identify malicious logic because of the absence of execution. It can be seen that multinomial malware detection by static analysis may give either non-accurate or very complicated decision models as described by Shalaginov et al. [381]. Therefore, we believe that behavioural characteristics can be a better option for multinomial malware classification. Furthermore, the application of Soft Computing methods as shown by Grini et al. [167], a field of Machine Learning (ML), can result not only in accurate, but also human-understandable models.

Static Characteristics of Windows PE Files

A literature study was performed with the intent of understanding how static malware analysis can create a synergy with Machine Learning methods to automate

detection. Since the field of Machine Learning is broad, many authors use their own unique set of methods to perform classification. Therefore, we first we focus on the common static characteristics and later go into the details of the methods and their relevance to different characteristics. The PE file format was introduced in Windows 3.1 as PE32 and further developed as PE32+ format for 64 bit Windows Operating Systems. PE files contain a Common Object File Format (COFF) header, standard COFF fields such as header, section and table, data directories and Import Address Table (IAT). Beside the PE header fields, a number of other static features can be extracted from a binary executable such as the strings, entropy, and size of various sections.

As outlined above, a number of works have been developed that use PE32 headers a basis for static analysis. Blount [82] explored the application of a rule-based classifier to extract fuzzy rules for malware detection on 3,401 malware and 3,373 goodware samples. It showed a strong performance of fuzzy rules based on PE32-based features. Markel et al. [264] used PE32 header data in malware and benign files detection by Classification Tree, Naive Bayes, and Logistic regression. Authors achieved a 0.97 F-score on binary classification. Furthermore, Shankarapani et al. [385] applied a PE file parser to extract static features for similarity analysis of trojans, viruses, adware, and spyware. Overall, 1,593 samples were acquired for binary classification. As for multinomial classification, there are few works worth mentioning. Zhang et al. [458] explored binary classification of 450 samples of the trojan and worm virus subsets against benign files using 2-gram analysis of binary values in PE file. Other work was done by Rieck et al. [333] who studied 14 different malware families extracted from 10,072 unique binaries. Authors achieved on average 88% accuracy of family detection using separate binary SVM.

To be able to apply Machine Learning to PE32 files, static characteristics should be converted into machine-understandable features. There are different types of features depending on the nature of their values such as *numerical* that describes a quantitative measure (can be integer, real or binary value) or *nominal* that describes a finite set of categories or labels. An example of the *numerical* feature is CPU (in $\%$) or RAM (in MB) usage, while *nominal* can be a file type (like $*.dll$ or $*.exe$) or API function call (like $write()$ or $read()$).

1. *n-grams of byte sequences* is a well-known method of feature construction utilizing sequences of bytes from binary files to create features. Many tools have been developed for this purpose, such as *hexdump* [268] which created 4-grams from byte sequences of PE32 files. The features are collected by sliding window of *n* bytes. This resulted in 200 million features using 10-grams for about two thousands files in overall. Moreover, feature selec-

tion was applied to select 500 of the most valuable features based on the Information Gain metric. Such features achieved an accuracy of up to 97% for malware detection. Another work on byte n-grams [332] described the usage of 100-500 selected n-grams on a set of 250 malicious and 250 benign samples. A similar approach [230] was used with $10, \ldots, 10,000$ best n-grams for $n = 1, \ldots, 10$. Additionally, ML methods such as Naive Bayes, C4.5, k-NN and others were evaluated for their applicability and accuracy. Finally, a range of 1-8 n-grams [203] can result in 500 best selected n-grams that are later used to train AdaBoost and Random Forests in addition to previously mentioned works.

2. *Opcode sequences* or operation codes are a set of consecutive low level machine abstractions used to perform various CPU operations. As was shown [366], such features can be used to train Machine Learning methods for the successful classification of malware samples. There should be a balance however between the size of the feature set and the length of n-gram opcode sequence. N-grams with the size of 4 and 5 result in the highest classification accuracy, as unknown malware samples can be revealed on a collection of 17,000 malware and 1,000 benign files with classification accuracy of up to 94% [354]. Also [84] explored the reliability of malware analysis using sequences of opcodes based on the 992 PE-files malware and benign samples. During the experiments, about 50 millions of opcodes were extracted. 1-gram- and 2-gram-based features showed good computational results and accuracy. Wang et al. [429] presented that 2-tuple opcode sequences can be used in combination with density clustering to detect malicious or benign files.

3. *API calls* are the function calls used by a program to execute specific functionality. We have to distinguish between System API calls that are available through standard system DLLs and User API calls provided by user installed software. These are designed to perform a pre-defined task during invocation. Suspicious API calls, anti-VM and anti-debugger hooks and calls can be extracted by PE analysers such as PEframe [57]. [452] studied 23 malware samples and found that some of the API calls are present only in malware, not benign software. Function calls may be composed in graphs to represent PE32 header features as nodes, edges and subgraphs [461]. This work shows that ML methods achieve accuracy of 96% on 24 features extracted after analysis of 1,037 malware and 2,072 benign executables. Furthermore, in [385], 20,682 API calls were extracted using PE parser for 1,593 malicious and benign samples. Such a large number of extracted features can help to create a linearly separable model that is crucial for many

ML methods such as SVM or single-layer Neural Networks. Another work by [352] described how API sequences can be analysed in concert with byte n-grams and opcode n-grams to extract corresponding features in order to classify malware and benign files. Also in this work, an array of API calls from IAT (PE32 header filed) was processed by Fisher score to select relevant features after an analysis of more than 34k samples.

4. *PE header* represents a collection of meta data related to a Portable Executable file. Basic features that can be extracted from a PE32 header are *Size of Header*, *Size of Uninitialized Data*, *Size of Stack Reserve*, which may indicate whether a binary file is malicious or benign [118]. Moreover, [410] utilized Decision Trees to analyse PE header structural information for describing malicious and benign files. [420] used 125 raw header characteristics, 31 section characteristics, 29 section characteristics to detect unknown malware in a semi-supervised approach. [431] used a dataset containing 7,863 malware samples from Vx Heaven web site in addition to 1,908 benign files to develop a SVM based malware detection model with an accuracy of 98%. [264] used F-score as a performance metric to analyse PE32 header features of 164,802 malicious and benign samples. [223] presented research of two novel methods related to the PE32 string-based classier that do not require additional extraction of structural or meta-data information from the binary files. [461] described the application of 24 features along with API calls for the classification of malware and benign samples from VxHeaven and Windows XP SP3 respectively. Furthermore, the ensemble of features was explored in [353], where authors used a total 209 features including structural and raw data from PE32 file header. Finally, Le-Khac et al. [243] focused on Control Flow Change over the first 256 addresses to construct n-gram features.

In addition to the study of specific features used for malware detection, we analysed articles devoted to application of ML for static malware analysis published between 2000 and 2016, which covers the timeline of Windows NT family that are still in use as depicted in the Figure 2.9. We can see that the number of papers that are relevant to our study grows significantly from 2009 on, which can be justified on the basis of the increase in the number of Windows users (potential targets) and corresponding malware families.

Challenges. Despite the fact that some of the feature construction techniques reflected promise precision of 90+ % in differentiation between malicious and benign executables, there are still no best static characteristic that guarantee 100% accuracy of malware detection. This can be explained by the fact that malware

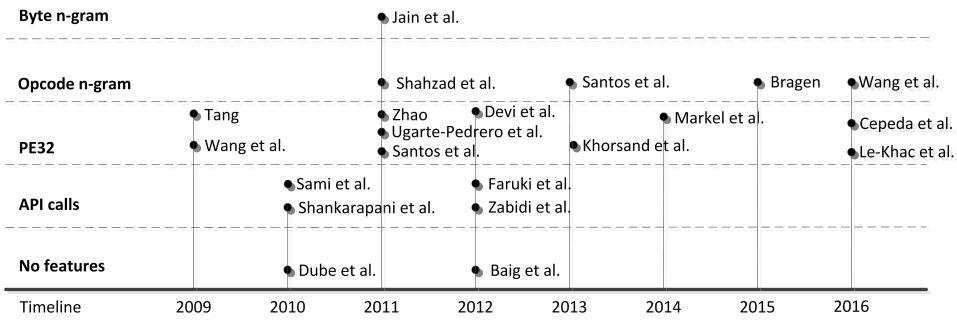


Figure 2.9: Timeline of works since 2009 that involved static analysis of Portable Executable 32bit files with respect to characteristics and ML methods for binary malware classification

are using obfuscation and encryption techniques to subvert detection mechanisms. In addition, more accurate approaches such as bytes N-GRAMS are quite resource intensive and hardly practical in the real world.

Taxonomy of Malware Static Analysis using Machine Learning

Our extensive literature study as reflected in Table 2.1 resulted to proposing a taxonomy for malware static analysis using machine learning as shown in Figure 2.10. Our taxonomy depicts the most common methods for the analysis of static characteristics, extracting and selecting features, and utilizing machine learning classification techniques. Statistical Pattern Recognition process [202] was used as the basis for our taxonomy modelling.

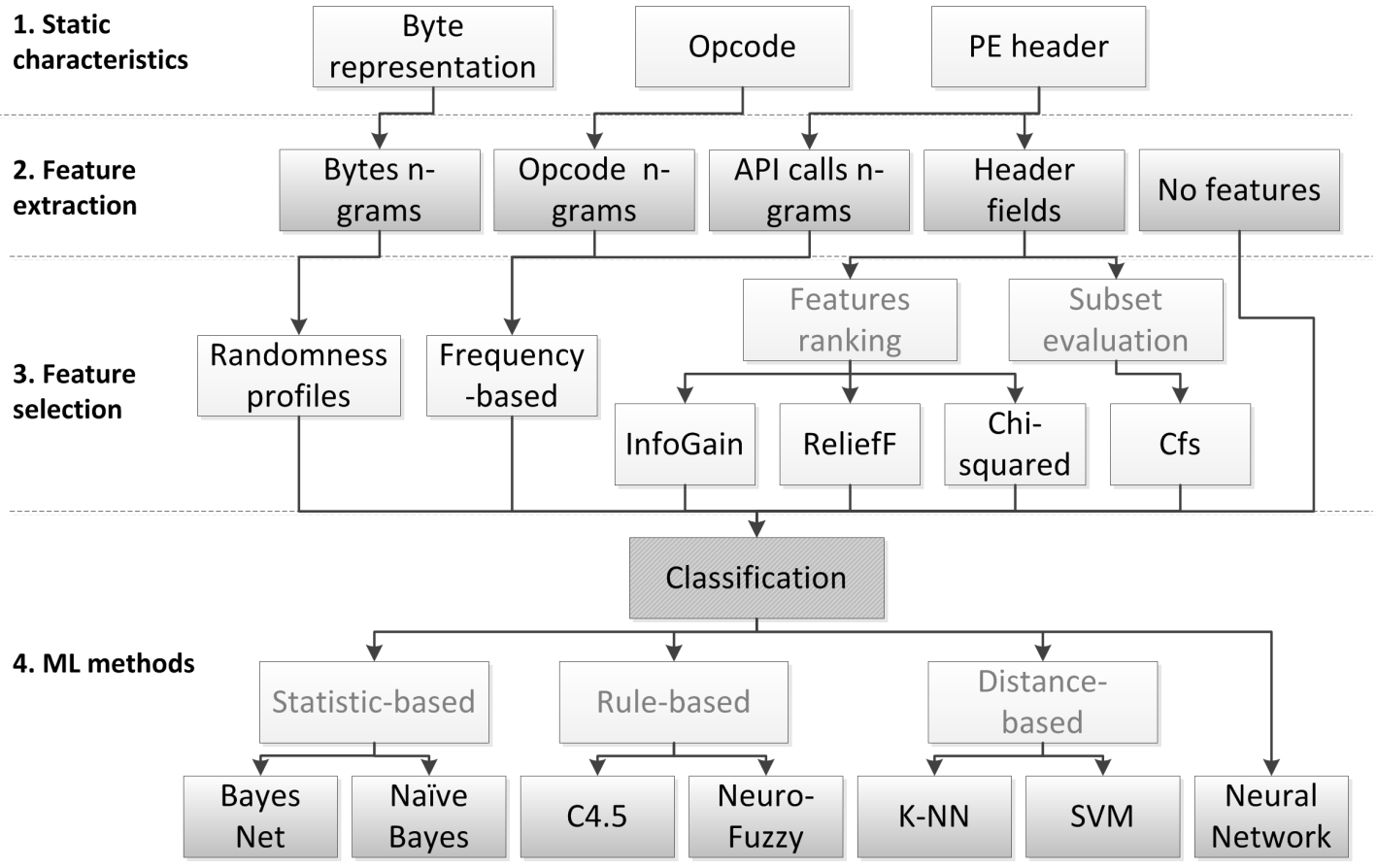


Figure 2.10: Taxonomy of common malware detection process based on static characteristics using Machine Learning

Year	Authors	Dataset	Features	FS	ML
			PE32 header		
2016	Cepeda et al. [90]	7,630 malware and 1.818 goodware	57 features from VirisTotal	ChiSq Selector with 9 features finally	SVM, RF, NN
2016	Le-Khac et al. [243]	Malicious: 94 ;Benign: 620	Control Flow Change and 2-6 n-grams	-	Naive Bayes
2014	Markel et al. [264]	Malicious: 122,799, Benign: 42,003	46 features use python 'pefile'	-	Naive Bayes, Logistic Regression, Classification and Regression Tree (CART)
2013	Khorsand et al. [223]	Benign: 850 EXE and 750 DLL; Malware: 1600 from VX heavens	eliminated	-	Prediction by partial matching
2012	Devi et al. [118]	4,075 PE files: 2954 malicious and 1121 Windows XP SP2 benign	2 + 5 features	-	BayesNet, k-NN, SVM, AdaBoostM1, Decision table, C4.5, Random Forest, Random Tree
2011	Zhao [461]	3109 PE: 1037 viruses from Vx Heavens and 2072 benign executable on Win XP Sp3	24 features from PE files using Control Flow Graph-based on nodes		Random Forest, Decision Tree, Bagging, C4.5
2011	Ugarte-Pedrero et al. [420]	500 benign from WinXP and 500 non-packed from Vxheaven; 500 packed + 500 Zeus	166 structure features of PE file	InfoGain	Learning with Local and Global Consistency, Random Forest
2011	Santos et al. [353]	500 benign and 500 malicious from VxHeavens, also packed and not packed	209 structural features	InfoGain	Collective Forest
2009	Tang [410]	361 executables and 449 normal trojan files	PE header structural features	-	Decision Tree
2009	Wang et al. [431]	Benign: 1,908, Malicious: 7,863	143 PE header entries	InfoGain, Gain raio	SVM
			bytes n-gram sequences		
2011	Jain et al. [203]	1,018 malware and 1,120 benign samples	1-8 byte, n-gram, best n-gram by documentwise frequency	-	NB, iBK, J48, AdaBoost1, RandomForest

2007	Masud et al. [268]	1st set - 1,435 executables: 597 of which are benign and 838 are malicious. 2nd set - 2,452 executables: 1,370 benign and 1,082 malicious	500 best n-grams	InfoGain	SVM
2006	Reddy et al. [332]	250 malware vs 250 benign	100-500 best n-gram	Document Frequency, InfoGain	NB, iBK, Decision Tree
2004	Kolter et al. [230]	1971 benign, 1651 malicious from Vx Heaven	500 best n-grams	InfoGain	Naive Bayes, SVM, C4.5
			opcode n-gram sequences		
2016	Wang et al. [429]	11,665 malware and 1,000 benign samples	2-tuple opcode sequences	information entropy	density clustering
2015	Bragen [84]	992 malware, 771 benign from Windows Vista	1-4 n-gram opcode with vocabulary 530-714,390	Cfs, Chi-sqaured, InfoGain, ReliefF, SymUncert.	Random Forest, C4.5, Naive Bayes, bayes Net, Baggin, ANN, SOM, k-nn
2013	Santos et al. [354]	13,189 malware vs 13,000 benign	top 1,000 features	InfoGain	Random Forest, J48, k-Nearest Neighbours, Bayesian networks, SVM
2011	Shahzad et al. [366]	Benign: 300, Malicious: 300 on Windows XP	coabulary of 1,413 with n-gram=4	tf-idf	ZeroR, Ripper, C4.5, SVM, Naive Bayes, k-nn
			API calls		
2012	Zabidi et al. [452]	23 malware and 1 benign	API calls, debugger features, VM features	-	-
2012	Faruki et al. [143]	3234 benign, 3256 malware	1-4 API call-gram	-	Random Forest, SVM, ANN, C4.5, Naive bayes
2010	Shankarapani et al. [385]	1593 PE files:875 benign and 715 malicious	API calls sequence	-	SVM
2010	Sami et al. [352]	34,820 PE: 31,869 malicious and 2951 benign from Windows	API calls	Fisher Score	Random Forest, C4.5, Naive Bayes
			no features / not described		
2012	Baig et al. [70]	200 packed PE and 200 unpacked from Windows 7, Windows 2003 Server	file entropy	-	-
2010	Dube et al. [130]	40,498 samples: 25,974 malware, 14,524 benign	from 32 bit files	-	Decision Tree

Table 2.1: Overview of PE32 malware analysis using static characteristics and ML methods

To get a clear picture on the application domain of each machine learning and feature selection method, we analysed the reported performance as shown in Figure 2.11. The majority of researchers were using byte n-gram, opcode n-gram, and PE32 header fields for static analysis while C4.5, SVM, or k-NN methods were mainly used for malware detection. Information Gain is the predominant method for defining malware attributes. Also, we can see that n-gram-based methods tend to use corresponding sets of feature selection (e.g. tf-idf and Symmetric Uncertainty) that are more relevant for a large number of similar sequences. On the other hand, PE32 header-based features tend to provide higher entropies for classification; therefore, Control-Flow graph-based and Gain Ratio are more suitable for this task.

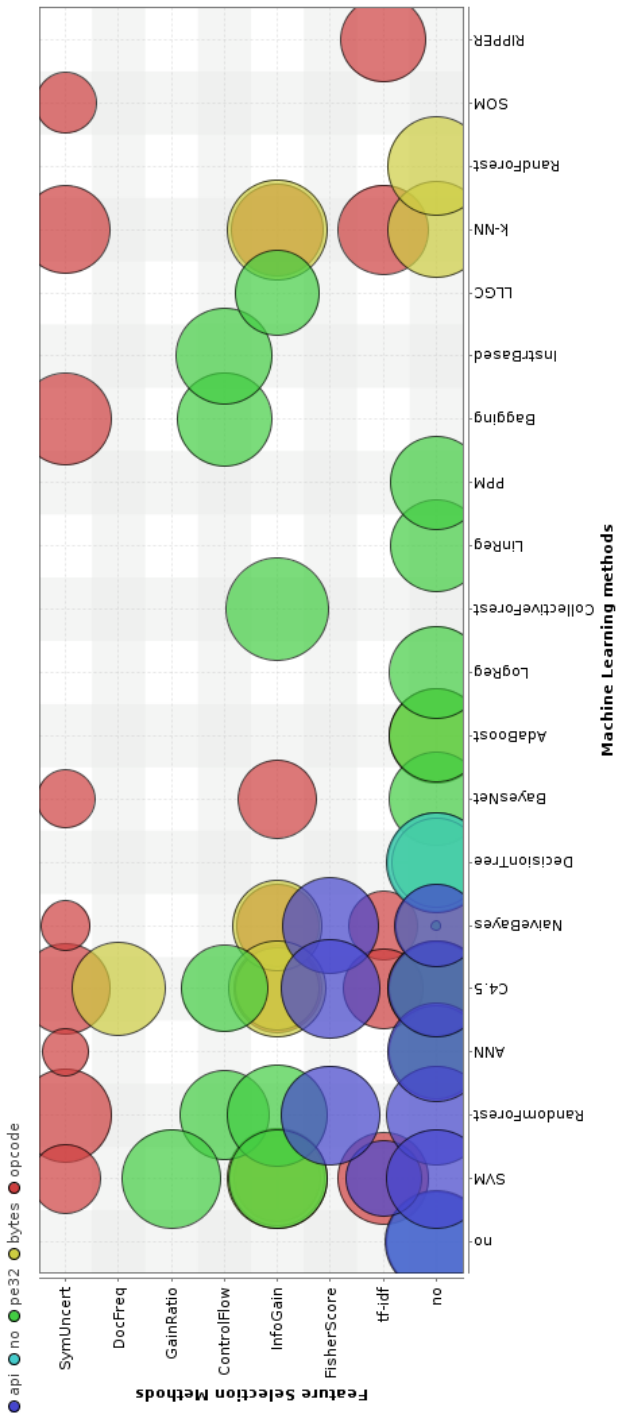


Figure 2.11: Comparison of accuracy of ML classification based on static characteristics with respect to feature selection. Colour of the bubbles shows characteristics used for detection, while the size of the bubble denotes the achieved accuracy

To conclude, one can say that the majority of authors either extract features that offers good classification accuracy, or use conventional methods like Information Gain. N-gram based characteristics need other FS approaches however to eliminate irrelevant features. Rule-based ML is the most commonly used classification method along with SVM. The forest-based method tends to be more applicable for PE32 header-based features. Also, ANN is not a commonly-used technique. While most work achieved an accuracy of 80-100%, some Bayes-based methods offered much lower accuracy, even down to only 50%.

From a review of the literature, we found that there is a no comprehensive research today that involves the categorization of malware into families or categories. Authors generally tend to mention a few common classes such as "Benign", "Virus", "Worm", etc. as described by Manehem et al. [276]. Despite the lack of study of such aspects, the authors of this work still emphasize the importance of discriminating between different malware types. This can be important when taking specific countermeasures against possible infection. Although some works consider only static characteristics [381], analysis of behavioural properties may yield good classification results. Rieck et al. [333] performed a study of 14 malware families, including *Trojan.Banker* and *Worm.SdBot* back in 2008. In it, authors considered the run-time observations of various activities of the files in CWSandbox. All these families correspond to only three malware categories (one Backdoor, two Trojans and thirteen Worms) despite a significant number of malware families studied. One can see that the dataset is also highly imbalanced, containing 1,500 samples of *Worm.Allapte* and the same number of *Worm.Virut* samples, while having only 91 samples of *Backdoor.VanBot*. Finally, the malware samples were examined in the beginning of 2007, which is almost 10 years ago. Windows Vista was released on the 30th of January 2007, meaning that the study included malicious software before this and later versions. Another work on behavioural malware analysis was done by Cheng et al. [98] and shows an application of a tf-idf scheme in malware families' identification. API call functions and their parameters (with values) were utilized as a main set of behavioural characteristics. Authors collected malware samples that belong to 2 malware types (Trojan and Worm) and 10 malware families. This work used only 425 malware samples, which were imbalanced in terms of the malware families' distribution (from 9 to 133 samples per class). A similar problem was investigated by Lin [252] in a Xen environment with the resulting classification of 9 malware families. One can see that the set was too diverse, naming W32, Suspect.Trojan and Trojan as various categories. Moreover, the authors also used a small corpus of benign samples from the System32 directory. On the other hand, when it comes to static analysis, the work done by Grini et al. [167] shows that many ML methods fail to classify multiple families. Therefore, we believe that the utilization of behavioural analysis may help to overcome

the limitations of a static approach. In conclusion, to our knowledge there has not been enough exploration of multinomial malware detection problems recently, especially in studying contemporary malware families and categories. Additionally, existing studies use either highly imbalanced or very diverse malware collections.

Multinomial Malware Classification[24]

Previous works mostly focus on the differentiation of a file between benign or malicious. This is a *binary classification*, where a heap of malware samples are classified against a collection of goodware. Cohen [104] suggested in 1987 that no algorithm will be able to confidently detect all computer viruses. This assertion was strengthened by Chess et al. [99]. As a result, we can assume that no methods can achieve 100% classification accuracy on large-scale sets. Bragen [84] applied Machine Learning (ML) on opcode sequences and achieved 95% accuracy with the RandomForest method. Kolter et al. [231] used 1,971 malicious files and 1,651 benign, while Bragen used only 992 malicious and 771 benign. Markel et al. [264] used PE32 header data in malware and benign files detection on Decision Tree, Naive Bayes, and Logistic Regression. The authors achieved a 0.97 F-score in binary classification. Furthermore, Shankarapani et al. [385] applied PE32 file parser to extract static features for similarity analysis. Overall, 1,593 samples were acquired for binary classification. The limitations of the afore-mentioned research are in the low number of files studied.

In contrary to binary, *multinomial classification* can be described as a detection of whether a malware belongs to a particular family or type. Rieck et al. [333] studied 14 different malware families extracted from 10,072 unique binaries. The authors achieved an average of 88% accuracy in family detection using an individual SVM for each one. Additionally, Zhang et al. [458] explored binary classification using binary sub-sets of 450 viruses and goodware based on the 2-gram analysis. Needless to say, not many works target the problem explored in this paper.

Most of the relevant works that can be found in the malware analysis community are devoted to the classification of a file into benign or malicious. This is a simple *binary classification*, where various collected malware samples are classified against a collection of goodware. Kolter et al. [231] used 1,971 malicious files and 1,651 benign to this end, while Bragen used only 992 malicious and 771 benign. Markel et al. [264] used PE32 header data in malware and benign files detection on Decision Tree, Naive Bayes, and Logistic Regression. The authors achieved a 0.97 F-score on binary classification. Shankarapani et al. [385] applied PE32 file parser to extract static features for similarity analysis. Overall, 1,593 samples were acquired for binary classification. The limitations of the aforemen-

[24]The ideas of this subsection are published under contributions [167, 381]

tioned researches are in the relatively low number of files.

In contrast to binary, *multinomial classification* can be described as a detection of whether a malware belongs to a particular family or type. There exist a number of *malware categories*, (trojan, backdoor, etc) and *malware families*, (Poison, Ramdo, etc), which are commonly defined by the Information Security community. A *malware category* is a general type of malware that uses a certain kind of approach to exploit a system and gain illegal access, such as a *worm*, which is a self-replicating code that can spread over email, or *ransomware* that encrypts files and requires a financial ransom to be paid [97]. On the other hand, a *malware family* is a specific sub-category that uses a particular vulnerability or targets specific software versions. For example, considering the *worm* category, we can distinguish the *p2p worm* family like *Spybot* from *removable drive worm* like *Autorun!inf* [11]. Cohen [104] suggested in 1987 that there are no algorithms that will be able to confidently detect all possible computer viruses. This statement was strengthened by Chess et al. [99]. Rieck et al. [333] studied 14 different malware families extracted from 10,072 unique binaries. The authors achieved on average 88% accuracy in family detection using individual SVM for each one. Further, Zhang et al. [458] explored binary classification using binary sub-sets of 450 viruses and goodware based on the 2-gram analysis.

Malware naming by anti-virus vendors

There exist a number of malware types (like *trojan, backdoor*, etc) and families (like *Poison, Ramdo*, etc), which are commonly defined by the Information Security community. In 1991, the Computer Antivirus Research Organization (CARO) proposed a standardized naming scheme for malware [4]. Although CARO states that this naming scheme is "widely accepted", we found that from all the vendors on *VirusTotal*, apparently Microsoft is the only one that complies with this. It is therefore challenging to establish a common pattern in scanner results across anti-virus databases. An example of CARO naming is given in the Figure 2.12.

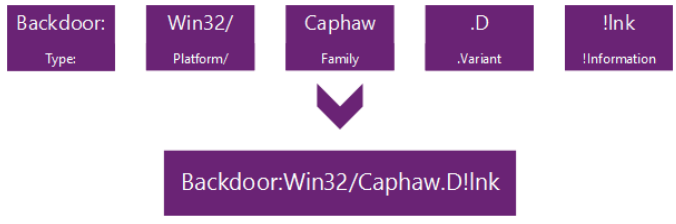


Figure 2.12: CARO malware naming scheme [281]

To the authors' knowledge, there has been no adequately performed comprehens-

ive study that provides a complete taxonomy of malware. However, there can be found blog entries with different malware species descriptions and dissections. Mushtaq [291] gave an overview of the top malware samples considering only 20 species, mostly families. Another comprehensive list of families is published by Microsoft as a part of the description of the Windows Malicious Removal Tool [280] starting from 2005 up until July 2016. VxHeaven also offers an overview of computer viruses. Finally, The Malware Database offers a large collection of different pages, also properly structured for each particular malware family and category [11]. Even in the scientific community, authors usually mix up both families and categories, and typically consider fewer samples than exist in in the wild [252, 333, 458].

Soft Computing in multinomial classification

For the classification of viruses, static features can be automatically extracted from PE32 headers and the results be used to build the classification model(s). To the authors' knowledge, the application of SC for multinomial malware detection has not been studied. With respect to classification problems, SC encompasses a set of powerful methodologies that can produce generalized models, although with inexact solutions. Some of the existing methods, such as MLP and SVM-based classifiers, were originally designed for binary problems. Other methods such as Naive Bayes and Bayesian Networks were designed to handle multinomial tasks. In a prominent study by Ou et al. [303], different models of multilayer ANN were studied with respect to multi-class problems. In a different paper by Shalaginov et al. [372], some thoughts regarding the application of Neuro-Fuzzy for multi-class problems were presented with a number of improvements. However, the data sets used were small.

Limitations of anti-virus solutions

Conventional signature-based anti-virus products ascertain specific system artifacts that malware leaves in a system. Such a method has a series of drawbacks however, including the inability to detect new malware or malware that uses polymorphic code. With a growing number of new malware categories and zero-day attacks, classical signature-based anti-virus software work is more challenging [288, 390]. This is caused by the fact that such software relies on static signature sets, which are maintained by the developer company. Moreover, according to [110], classical signature-based antivirus solutions fail to consider the challenges of encryption, polymorphism, and other obfuscation methods. Another problem which appears after signature composition is the complexity of the signatures themselves, as well as the huge amount of various signatures sets. Such problems can be approached by ML and behavioural analysis. There exist a num-

ber of so-called sandboxes and on-line services for dynamic malware analysis such as Cuckoo, cwsandbox, VirtusTotal, Malwr, etc. However, these tools require file submission to a third-party server or installation of a specifically-crafted system. Our goal is to speed up malware detection using observed behavioural features on a common MS Windows installation.

2.4.2 Network Intrusion Detection

Almost all devices with computational capabilities are now connected to the Internet, including PCs, mobile phones, and even IoT. As a result, there might be a number of ways attackers can exploit different vulnerabilities to gain access or perform other illegal activities. A single wire establishes connection between a computer system and Internet. A commonly accepted Open Systems Interconnection model (OSI model) offers the standardization of network communications without taking into consideration the used architecture. Seven layers of the OSI model are depicted in the Figure 2.13, and show general attack vectors against a computer system in the network.

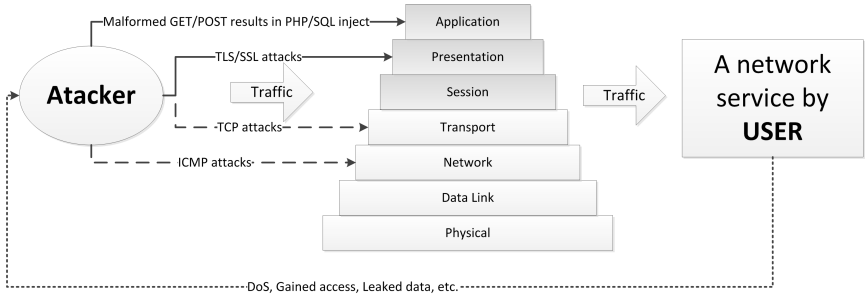


Figure 2.13: A general example of approaches used in network attacks in the Internet

The nested layers structure of OSI model includes (down-up): Physical, Data Link, Network, Transport, Session, Presentation, and Application layers, according to an explanation by Zimmermann [463]. At the same time, each layer may include vulnerabilities or specific cases that lead to particular attacks being successfully carried on. CERT in USA described common examples of DoS attacks, their potential impact, and possible ways of mitigating such attacks [29]. At the same time, SANS explained such attacks by providing examples and a justification of the situation being examined [405].

Network Forensics is a field of Digital Forensics that targets analysis and evidence extraction from network traffic. Considering the increase in bandwidth and amount of traffic travelling in networks over the last decades, much research has been done on the application of Computational Intelligence to decrease the amount

of required manual work as described by Muda et al. [289]. In 2014, Al-Mahrouqi et al. [49] presented a Network Forensics readiness and security awareness framework. We can see based on that research that it is not only important to preserve data properly, but also to analyse it meaningfully and extract relevant knowledge. Adeyemi et al. [44] studied features that are relevant to network forensics investigations with respect to different stakeholders. One can see that there are many comprehensive characteristics that can be helpful in differentiating between normal traffic and attacks.

Fuzzy Logic in Intrusion Detection[25]

Since the number of data that transfers through a network has increased significantly over the last decade, many researchers suggest using Computational Intelligence for network traffic analysis. This may partially eliminate the need for manual analysis, which can be a major limitation. There are a number of works in the area of network traffic detection and classification. Singh [392] studied the performance of unsupervised methods on network traffic detection such as k-means and Expectation Maximization. In general k-means has much better accuracy in the detection of DHCP, SMTP, ICMP, HTTP and DNS traffic considering different features of network packets. These are flow duration, packet length, total number of packets, and total number of bytes transferred. Using a higher level OSI model, one can detect SQL injections according to Makiou et al. [261]. HTTP dissection is used and the content is then compared against security rules for both detection and prevention of the attacks against web servers.

Machine Learning methods have been studied in many works and have proven their effectiveness. Senel-Kleine et al. [361] studied SVM and Decision Trees (DT) on 20 different sets of network traffic and showed accuracy of up to 99% on anomaly detection. The authors stated however that to improve this result, other methods have to be applied to get better detection results from other sources. In addition to SVM and DT, Zhao et al. [460] also explored the Naive Bayes classifier. All three methods give a an independent high detection accuracy of real-time anomaly detection from a set of used features. Without destination and source IP however, it is hard to achieve the best possible accuracy. Since SVM is one of the best classifiers, Hong et al. [187] performed a study of how the SVM can be iteratively tuned to anomalies in the traffic. The proposed method has the best Accuracy/Time coefficient in comparison to standard configuration. Moreover, Singh et al. [393] also presented a survey of Machine Learning for Intrusion Detection Systems (IDS), which can cover not only attacks from outside the system, yet also from inside. In particular, authors investigate Neural Networks and Fuzzy Logic. It was stated that

[25]The ideas of thi sub-subsection are published aunder contributions [375, 377]

the main drawbacks are overfitting and high resource consumptions respectively. However, the synergy of these two models can bring a generalized rule that overcomes previously mentioned constraints. In addition, Zamani et al. [456] utilized KDD CUP 99 dataset to show the utility of Machine Learning methods for IDS, and concluded that they conform to the requirements of efficient systems designed to detect attacks in the networks.

Furthermore, some authors concentrated specifically on the application of the NF approach for network security. Shafiq et al. [362] studied how the Adaptive Neuro Fuzzy Inference System is applied for portscan detection. They used multiple worms with a variety of UDP/TCP ports. However, the authors also used the Takagi-Sugeno model, which is not entirely suitable for classification, but rather for regression purpose. To further elaborate, Aguiar et al. [45] studied how the Linux *Netfilte* / *Iptables* firewall can be fuzzified with respect to DOS protection using fuzzy rules. The authors believed that NF has a great potential for this and, in particular, they applied SOM to extract 4,800 clusters that characterise the set of 150,000 9-dimensional tuples. Recently Nguyen et al. [295] developed a NF model for online phishing detection with accuracy of up to 99.10% based on six input characteristics. Another work done using KDD Cup 99 set was by Amiri et al. [58] was targeted on dimensionality reduction to overcome Curse of Dimensionality. The number of features was reduced to 5 from 41 original while accuracy was still very good. However, to the authors knowledge there have not been studies done on the trade-off between accuracy and interpretability in the fuzzy inference model for firewall application. Moreover, the challenges with large-scale analysis have not been raised with respect to NF overfitting by large number of clusters from SOM clustering. Thus, we can see that the properties of network packets can be used to detect different types of traffic and to extract classification rules as result.

Real-World Application Scenario

Modern firewalls analyse traffic using different OSI model layers as described by Woodall [443]. We can see that this is a dynamic area with continuous changes in attack vectors, data intensity and technologies. In most cases the filtering rules are used to detect whether a traffic packet falls into category of attack or normal. From the literature review we can see that rules generated by Machine Learning methods can play an important role in Network Security. As a real-world example of the firewall using fuzzy rules, we suggest using the following scheme depicted in the Figure 2.14. It includes following components: a reference set of known-to-be malicious and benign network traffic to learn the model, a NF engine capable of fast training, and the ability to produce a generalized model and a pool of fuzzy rules used by firewall. This is however a decision support system, so the system

administrator has to give his analysis and tune the model when necessary.

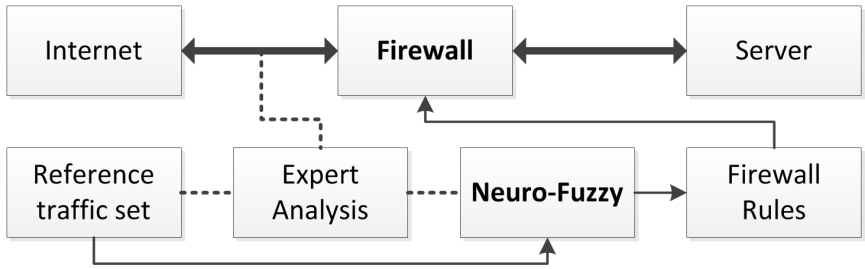


Figure 2.14: Example of Neuro-Fuzzy application in Network Firewall

Fuzzy Logic can help to eliminate multiple problems that exist in cases with Hard Computing methods. This is due to the fortification and a level of abstraction that helps to tune the parameters of the rules without the need for a complete retraining of the model. Moreover, inference from such rules is not going to take much time and can be performed using parallel processing and RAM storage. We believe that presented architecture can be beneficial for large organizations and can reduce amount of efforts necessary for firewall maintainance.

2.4.3 Application Level Security and Attacks on Web[26]

Over the last couple decades, web applications became a popular means of service delivery to customers. Recently, web applications have come to be considered as a part of Cloud Computing meaning, that the data and computing power do not need to be on the client's side. This makes the utilization of the web extremely convenient and suitable for a number of domains, including Information Security services. Yet despite the fact that we can protect against multiple attacks on network protocol level, this makes it more on the application level. As was studied by Shah et al. [364] a typical web application setup includes a number of components like web servers, databases, plugins, etc. Attacks on these parts cannot be prevented by standard network firewalls. Scenarios of attacks were given by the OWASP Top Ten project [305]. Prandl [319] studied web application firewall (WAF) solutions such as ModSecurity, WebKnight, and Guardian. As can be seen, these solutions have different performances considering malicious and benign HTTP traffic. Despite the fact that ModSecurity can be considered as one of the best open source solutions, it is designed to be used only with a small number of web servers such as Apache, IIS, and Nginx server. It makes the utility of WAF narrow and not scalable to other software. Prokhorenko et al. [323] performed a comprehensive analysis of web application protection techniques. It clearly shows that the major-

[26]The ideas of this subsection are published under the contributions [372, 378]

ity of research targets the aspect of web applications, where a majority of attacks happen due to improper data flow handling and filtering. From the other side, verification of the input is one of the most common protection techniques considering the possibility to analyse it directly from HTTP request. Finally, there number of attacks on web applications is large, and it is hardly possible to produce signatures for HTTP traffic that can detect possible obfuscations.

Conventional signature-based rules do not offer as much flexibility as fuzzy rules. We can see that fuzzy rules also might be beneficial when there are not much statistics describing an attack. Kadiervelu et al. [210] studied how fuzzy logic can be applied to detect unknown attacks against web applications. Authors proposed C4.5-based fuzzy intelligent system to detect anomaly SQL queries. Further, Geraily et al. [162] used Hidden Markov Model as an input to fuzzy inference to detect malicious HTTP requests. Shiaeles recently wrote a thesis [389] on real time detection of DDoS in web services. The authors developed a fuzzy-based detector for spoofing attacks, which were predicted based on the User Agent identity, HTTP requests, etc. Moreover, the limitations of popular KDD CUP 99 data was described since it does not contain information about the user agent and visited URL. Another work done by Atienzaet et al. [64] explicitly described the application of Neural Networks architectures for attacks detection in HTTP traffic using CSIC HTTP 2010 dataset. In particular, authors used unsupervised learning by SOM for similarity grouping of normal and abnormal HTTP requests.

The mentioned works concentrate only on the binary classification of benign and malicious traffic rather than on detection of attack type. Also, we can see that fuzzy rules have great potential when it comes to detection of attacks, however multinomial classification of the type of web attack was not well studied, according to the authors knowledge.

The advantage of fuzzy rules is that they can describe each group of similar data that belongs to different classes separately without designing a separation hyperplane, as is normally done in other binary classification methods. Considering also the area of Digital Forensics, such a model underlines the great interest in building human-understandable and interpretable models that are also accurate. The majority of the tasks in illegal activity detection denote either *"malicious"* or *"benign"* patterns, which is a binary classification task. These can be software samples, network traffic dumps, web pages, etc. There are studies however that require the determination of a specific group or domain which this *"malicious"* pattern belongs to, for example network attacks or malware families.

Many methods such as Multilayer Perceptron (MLP) or Support Vector Machine (SVM) were originally designed to deal with a binary output in a form acceptable

to computers at the dawn of the ML era. The design of MLP as Neural Network is purely based on the activation function (e.g., logistic) that either activates (state "1") or deactivates (state "0") the output neuron. There is almost no way to use such single-output NN architecture for multinomial classification. Thus, one must utilize either one network with multiple outputs or multiple single-output networks for the defined purpose according to Aly [56]. Multiple outputs are normally used for the explicit determination of class label or per class probability. Ou et al. [303] performed an extensive study of the multinomial classification by NN and concluded that the optimal class boundaries can be found when the method separates all classes at the same time by a single NN model.

2.4.4 Network Forensics Readiness

Due to the growth of network bandwidth and the development of new attack scenarios, Network Forensics faces multiple challenges related to large-scale dataset processing and evidence extraction, as mentioned in the report by Ernst & Young [140]. It is important not only to detect an anomaly and classify it as a likely attack, but also to provide human-understandable Threat Intelligence through a corresponding statistically-based analysis. Conventional Computational Intelligence methods are no longer reliable as they either result in a very complex model that is hard to understand or take too long a time to infer a meaningful model.

Computational Intelligence in Network Forensics[27]

Network Forensics is a field of Digital Forensics that targets the analysis of and evidence extraction from network traffic. Considering the increase in bandwidth and amount of traffic travelling in networks over the last decades, much research has been done on the application of Computational Intelligences to decrease the required amount of manual work, as described by Muda et al. [289]. On the one hand, authors use FL as a prominent approach; e.g. Vural et al. [428] shows how the bot-nets can be detected using FL, where email changing behaviour was used as an output of the fuzzy system and then fed into the network forensics analysis module. Another Fuzzy Logic-based Expert System was proposed by Kim et al. [226] to facilitate the automated analysis of evidence in network traffic. The performance of this system reached 93% on 1998 DARPA dataset. Recently, Rostamipouret al. [340] also studied the application of FL for Network Forensics, in particular the detection of the origins of buffer overflow attacks.

On the other hand, many works target the application of different types of ANN. Huang et al. [190] suggested using growing hierarchical SOM, as it can help identify different patterns of DDoS attacks successfully. The same authors used

[27] The ideas of this sub-subsection are published under the contribution [376]

SOM for anomaly detection in concert with SVM to explore the patterns of anomalous network packets [191]. Another application of SOM is in the visualisation of Network Forensics Traffic Data, as suggested by Palomo et al. [308]. The authors used manually defined sizes of SOM from 3x3 up to 7x7 on a set of 150,871 samples. Also worth mention is the work presented by Yan [447], where ANN was applied on 10% KDD CUP 1999 dataset over a range of features such as running state of host, communications of network, and content control domains.

Considering previous works in this area, we can see that both FL and ANN (SOM in particular) are widely applied to detect anomalies as possible attacks. Most authors don't apply any Hybrid Intelligence however, and extract the parameters of fuzzy rules mostly manually. To mitigate this limitation, we target Neuro-Fuzzy that uses an unsupervised SOM method to extract these parameters without human interaction. To the author's knowledge, there is a single relevant research available on Neuro-Fuzzy for Network Forensics done by Anaya et al. [59] to detect suspicious flows based only on TCP/IP LAN that have been compromised. However, this work mentioned neither the details of Neuro-Fuzzy architecture used nor the specific fuzzy sets used for detection. We can assume, however, that the authors used manually-crafted FL parameters.

Digital Forensics Readiness and Information Security Risk Management[28]

Another aspect of Digital Forensics Readiness is Information Security Risk Assessment (ISRA). According to Rowlingson [341] Network Forensics Readiness should be an inseparable part of the ISRA in order to be able to predict possible misbehaviour and plan corresponding mitigation actions. Furthermore, using historical information and statistical modelling, one can enhance Cyber Threats Intelligence and pro-active digital forensics for better preparation against cyber attacks and more efficient cybercrime investigations [168, 403].

Information Security and Risk Assessment. ISO/IEC 27005:2008 defines information or ICT risk as *the potential that a given threat will exploit vulnerabilities of an asset or group of assets and thereby cause harm to the organization.* Probabilistic risk analysis (PRA) is the preferred approach to risk in information security, where impact to the organization (e.g. loss if a risk occurred) and probability calculations of occurrence express risk. There are no standardized statistical approaches to information risk; to calculate risk (R) we apply the definitions provided by Aven [66] (p.229) for discussion and risk calculation. Where risk is described by events (A), consequences (C), associated uncertainties (U) and probabilities (P). U and P calculations rely on background knowledge (K). Also, model sensitivities (S) are included to show dependencies on the variation of the assump-

[28]The ideas of this subsection are published unders the contributions [432, 433]

tions and conditions. Thus, *R=f(A, C, U, P, S, K)*. A quantitative risk assessment in this sense derives from applying statistical tools and formal methods, mainly based on historical data (e.g. the law of large numbers), obtained distributions, and simulations. Thus, based on the definition of risk by Aven, we will consider applications of relevant methods for quantitative risk evaluation in terms of R. A risk assessment is very seldom purely quantitative, as there are assumptions *K* underlying the forecast. Finally, exposure is a crucial concept in risk management that we define as the susceptibility of an organization to a particular risk.

Statistical methods for historical data analytic in ISRA. One makes a decision about information security risks mostly based on previously collected data within the company or based on the publicly available historical data about causes and results [219]. We introduce several community-accepted methods to deal with historical data and be able to make quantitative risk assessments possible since qualitative risk assessment has precision limitations when it is necessary to make predictions in numbers.

Probabilistic modeling. This type of analysis is applied when there is need for probability estimation of a particular event x occurrence in a given historical dataset. Initially, the model $p(x)$ is built, and gives an estimation of the corresponding set of parameters from the data [164]. Then, this model can be used to estimate the probability of similar events in this very period or later on. We can state that there are many obstacles related to probabilistic modelling. First, very few data points from history may precipitate a wrong decision. Second, very rare events, like in the case of Fourth Quadrant, have negligibly small probabilities. This does not mean however that this event are not going to happen.

Numerical analysis. Numerical analysis is a broad field of data modeling, time series in particular. The function $f(x)$ is built using a previous period of time $x_0, \cdots, x_t$. To construct a proper model, available historical data must be decomposed into trends, seasonal components, and noise in order to build a precise prediction model. At this point, the recent data should possess a higher degree of trust than data from a long time before [61]. For the defined earlier research questions that statistical models can be applied to support risk assessment within the four quadrants, yet under some limitations, we consider the following supplementary statistical approaches [61]:

1. *Logistics function* describes the process where the initial impact causes an exponential increase until some moment in time. After this moment, growth will decrease until it is saturated to some ceiling value. [119].

2. *Conditional Probability* and Bayes Theorem are the probability methods

used to calculate the likelihood of the occurrence of some event when another dependent or independent event has already happened.

3. *Gamma distribution* represents a family of continuous probability distributions that can describe data with quite various characteristics. The main parameters are shaped k, and the scale of the distribution θ.

4. *Exponential growth* characterizes an event that does not have an upper boundary, and the observed outcome will grow more during the next period in comparison to previous.

5. *Log-normal probabilistic model* defines the distribution of some historical data under the condition that the logarithm of the data follows the Gaussian distribution.

From our point of view, these methods are the most promising for estimation of possible event outcomes based on previously analysed information.

Statistical hypothesis testing. Furthermore, we will justify the usage of specific statistical methods for each case study and make a hypothesis about their applicability in that particular case. At this point, we need to use statistical tests to verify the suggested hypothesis[29]. The two following approaches can be applied with probability distributions: QQ-PLOT, a Quantile-Quantile plot representing a probability plot by depicting expected theoretical quantiles E and observed practical quantiles O against each other, and STATISTICAL TESTS that estimate the quantitative metrics of how well the data fits hypothesized distributions.

Confidence Intervals or CI relates to the probabilistic estimation of whether a particular data or data sample is being placed within a hypothesized distribution. It also means that the defined in CI % of data will be in the hypothesized distribution. To be precise, the test evaluates the actual observed data O with the expected data E from the hypothesized distribution.

Fuzzy Logic in Risk Management. Most risk models are developed relying on the probabilistic data that derived from historical observation in additional to empirical measures. The main drawback of such models is the difficulty in analysing risk and making the corresponding predictions when the number of historical data is insufficient or unreliable. On the other hand, manual analysis is required to map specific probabilities to qualitative linguistic measures. Fuzzy Logic can be considered a solution to these problems since it may reduce the analysis complexity while providing more meaningful models to decision makers. The main idea not

[29] http://www.ats.ucla.edu/stat/stata/whatstat/whatstat.htm

only to give answers in terms of probability, as in classical probabilistic models, but to describe cause-effect relationships and bring situational awareness that may be useful in future intelligence. Shang et al. [384] explained in detail how fuzzy logic can be applied to both Qualitative and Quantitative analysis. Though study did not include aspects of fuzzy rules inference. Then, their colleagues [387] presented an in-depth study with applications of a large variety of fuzzy logic models, including fuzzy rules. Furthermore, Takacs [409] made an overview of the fuzzy rule-based system for Risk Management. The author highlighted the scalability of such an approach, and even the development of hierarchical systems for Risk Management. Another significant work that pays attention to fuzzy rules is done by Nunes et al. [298], where the authors highlighted the importance of fuzzy expert systems in handling value and incomplete data, and assisting human experts. Even diseases was discussed as one of the major possibilities for application. Oad et al. [299] performed a study on fuzzy rules for heart disease, fuzzifying many conventional measures like age, blood pressure, etc. The system shows performance of at least 80% in comparison to other, more complicated models. Generally speaking, fuzzy logic introduces the fundamentally new concept of possibility in the event in Risk Management along with conventional likelihood-based estimators.

2.4.5 Mobile Devices Malware

Significant development in mobile devices has occurred during the last decade, as average mobile phones have become powerful intelligent devices capable of making intense computations, comparable to personal computers and laptops. The absolute majority of mobile devices have installed Google Android OS due to the fact that it is open-source and allows users to easily install third-party applications. Another point is that most consumer electronics companies can now manufacture phones and install their own version of Android. As a result, the Android market share reached 81.7%, while Apple iOS is 17.9%, while both platforms make it an overall 99.6% of the world usage of mobile devices by the end of 2016 [426]. Other platforms share a considerably lower number of devices using them, e.g. Windows Mobile – 0.3%.

On the other hand, the ability to install third party applications on Android OS makes it more vulnerable to malware, which is additionally helped by the fact that user has power to accept all requested permissions. As a matter of fact, such framework gives an attacker the freedom not only to use financial services, but also to steal private information [365]. Following the growing amount of malicious software and threats, Android malware analysis has become a popular topic in Digital Forensics as well as in the Machine Learning community [48, 92, 116, 205, 246].

2.4.6 Privacy Preserving and Access Control[30]

AC is intended to limit access using different policies and models and to prevent an unauthorized access. Sahafizadeh et al. [345] provided a comprehensive overview of modern AC models. It studied Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), etc. MAC relies on levels of authorization or class of a user when evaluating its ability to perform operations on the object. DAC provides a specific detail for each particular user or user role of where she is capable of going. RBAC uses an access matrix of objects and subjects in order to define an access rule. ABAC [3] is one of the initiatives from NIST towards moving from Role-Based Access Control to a more flexible evaluation of the asset's attributes. Vincent [188] in the NIST guide defined a scenario that includes access control policy, environment conditions, and Subject and Object with corresponding sets of attributes. Similarity-Based Access Control (SBAC) as defined in the patent by Farber et al. [142] allowed the use of similarities in access attributes. Considering this, we can say that the application of Soft Computing can facilitate AC and is capable of learning on-line, since off-line may require significant resources when the organization size is large. There have also been some attempts to apply Machine Learning methods. The report [27] from NIST proposed a Risk-Adaptive Access Control model (RAdAC) that uses historical records to determine whether access should be granted or not. In addition, it was noted that Machine Learning, Evolutionary Computing (EC) in particular, can be included in RAdAC to improve the model. Furthermore, Bedi et al. [74] explored a way of applying ANN in the access of grid resources. In particular, requests for resources are classified via Radial Basis Function Neural Networks, because of non-linear ANN's superior generalization. However, to the author's knowledge, ANN has not been widely used in the AC models.

When talking about Mahcine Learning, Similarity-Based AC is affected by velocity and veracity of the access traces appearing in access logs. It means that the data are available for a short time frame and should be processed in a fast *on-line* way, rather than an iterative *off-line* [336]. *Data streams mining* is a special field that defines such on-line models [446]. From the perspective of Information Security, be it events monitoring, traffic processing, etc. logs access in addition to analysis [122]. Thus, the challenge with data stream mining can be formulated as follows: At time t_i, there happens some non-deterministic event T_i such as entering user credentials to access the resource, which also can be influenced by some covert action or can be completely random. Each user X_i can be described by a set of M properties (features) $X_i = \{A \in R^M\}$, where features $a = \{a_0, \cdots, a_M\}$

[30]The main ideas of this subsection are published under the contribution [370]

can be either user- or resource-specific. So, the goal is to predict the class Y_i of this event, which defines the actions to be undertaken (*"allowed"* or *"blocked"*). This has to be done using previously collected logs and established access policies over some past time t, as is shown in the Figure 2.15.

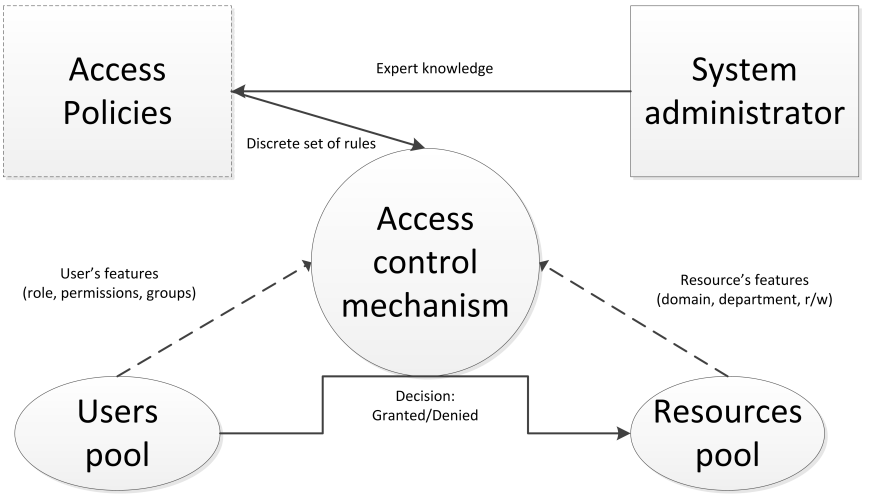


Figure 2.15: How the Access Control mechanisms generally interact with objects and subjects, according to ABAC [188]

In real world tasks, the statistics are so numerous that it is not possible to re-learn the model each time a new event T_{i+1} arrives. From this point of view, we concentrate on the data samples that have some predefined set of features A, where each feature is a numerical $\forall a_j \in A : a_j \in R$. The values of each feature a_j are unknown beforehand. Similarly, the combination of the features in the given data sample $X_1, \cdots, X_n$ in this one are from an access control system sample that has not yet appeared. Moreover, there is a need to determine a class Y_i of the given sample $X_i|_{i=1,\cdots,N}$, where N is the size of the training data.

2.5 Neuro-Fuzzy – A Hybrid-Intelligence Analytics

In this Section, we describe why it is important to synergize SC methods to overcome the limitations of stand-alone ones, and what kind of optimization can be applied to facilitate it. The crucial property of Soft Computing (SC) methods applied in malware detection is the interpretability of the model [394] used to explain and interpret found evidence. Fuzzy Logic offers a great trade-off between the accuracy of the model and its interpretability. NF is a Hybrid Intelligence method that combines both FL and ANN. Over last decades there have been developed multiple Fuzzy Inference Systems (FIS).

Generally, we can consider the two following FIS based on the nature of fuzzy rules [220]:

Mamdani is a type of fuzzy inference system that produces human-understandable answers to defined problems by assigning data to one or another linguistic fuzzy set with the possibility of retrieving the numerical measure of the assignment. Mamdani is classification-based, represented by Fuzzy Associative Memory (FAM), etc [39]. The Mamdani-type fuzzy rules look like following:

$$IF \;\; x \in A \;\; AND \;\; y \in B \;\; THEN \;\; c \in C_i \tag{2.3}$$

where x and y are numerical variable inputs that go through the process of fuzzification; A and B represent fuzzy sets and c is an output fuzzy variable that takes class label C_i as its value.

Takagi-Sugeno, on the other hand, is a Fuzzy Inference System that generates a crisp output to a system with fuzzy variables as input. Takagi-Sugeno is considered to be a regression-based system, represented by Adaptive Neuro-Fuzzy Inference System (ANFIS), etc [39]. This means that with this system, there is no need to perform defuzzification as in case with Mamdani-type rules. Such an approach however does not classify a data sample, but computes values, similar to regression trees. The fuzzy rule has the following view:

$$IF \;\; x \in A \;\; AND \;\; y \in B \;\; THEN \;\; c = x_i + y_i \tag{2.4}$$

where c represents a numerical variable that depends solely on the input variables x_i and y_i. In this work, we concentrate on Mamdani-type fuzzy systems, since the idea is to classify different activities and found information into crisp classes.

To enhance parameters extraction in Fuzzy Logic and improve the understandability of Neural Network, a group of Neuro-Fuzzy methods were proposed to be able to generate intelligent models. Neuro-Fuzzy (NF) is one of the most-commonly used methods for fuzzy rules construction. According to the taxonomy by Kruse [235], we can name following architectures:

Cooperative denotes a system that has neural networks and fuzzy systems, where the operation of one does not depend on the other. This means that neural networks either can be trained from data and provide rules parameters or extract fuzzy set parameters that are then later put into a pre-defined fuzzy inference system.

Hybrid in contrast to a cooperative model, this model incorporates a synergy of ANN and FL that results in an inseparable system that is trained simultaneously and may be used as a fully integrated classifier that gives not only a classification result, but also explains the data learned by ANN.

The original method described by Kosko [233] consists of two stages: the allocation of fuzzy rules parameters by means of Self-Organizing Maps (SOM), and tuning by means of Artificial Neural Network (ANN). NF learns automatically from data and produces human-understandable classification models based on fuzzy logic [54] similarly to different adaptive techniques like Fuzzy Adaptive Learning Control Network (FALCON) and Adaptive Network Based Fuzzy Inference System (ANFIS) [39]. These systems use self-organizing methods to define the initial parameters of the clusters used in the models.

2.5.1 Optimization for Large-scale Data Analysis[31]

From what we see in the literature, the fuzzy systems are in most cases the most important components of the SC models. The main point of SC for the end user is whether the decision made is understandable and explainable, and whether automated processes based on decisions are reliable. Following groups of users can be named: *Unaware* (very limited knowledge), *Limited awareness* (specific and basic usage of the systems), *Experts* (broad understanding of work principles). The margin of the solutions given by SC methods allows one to involve different factors in balancing between the different decisions.

There were two main aspects that we considered when performing the literature review. First of all, hybridization means that several SC methods are crafted to complement one another, resulting in a better-performance model. Second, metaheuristic optimization can support building an intelligence system as a way of performing multi-criteria optimization of complex tasks without human expert intervention.

The availability and labelling of the data used for model training affects the learning paradigm to be used in the model as well. With respect to the structuring of the data, we can distinguish *supervised*, *semi-supervised*, and *unsupervised* methods as described in the book [177] by Han et al. From the other side, the methods can be trained *off-line*, *on-line* incrementally, and as a *batch*. Furthermore, the models can be *linear* and *non-linear* that affects the complexity or required level of the abstraction of the model. However, all SC methods may help to find a human-understandable explanation or tentatively locate the decision as it is depicted in the Figure 2.16. It also shows how the different aspects may influence the construction of the DSS.

Despite the unique applicability of each method, there are disadvantages that limit utilization. Therefore, we consider the creation of SC-based Hybrid Intelligence models as the core consent for the decision support system in Crime Investigation

[31] Ideas of this subsection are published under the contribution [367, 368]

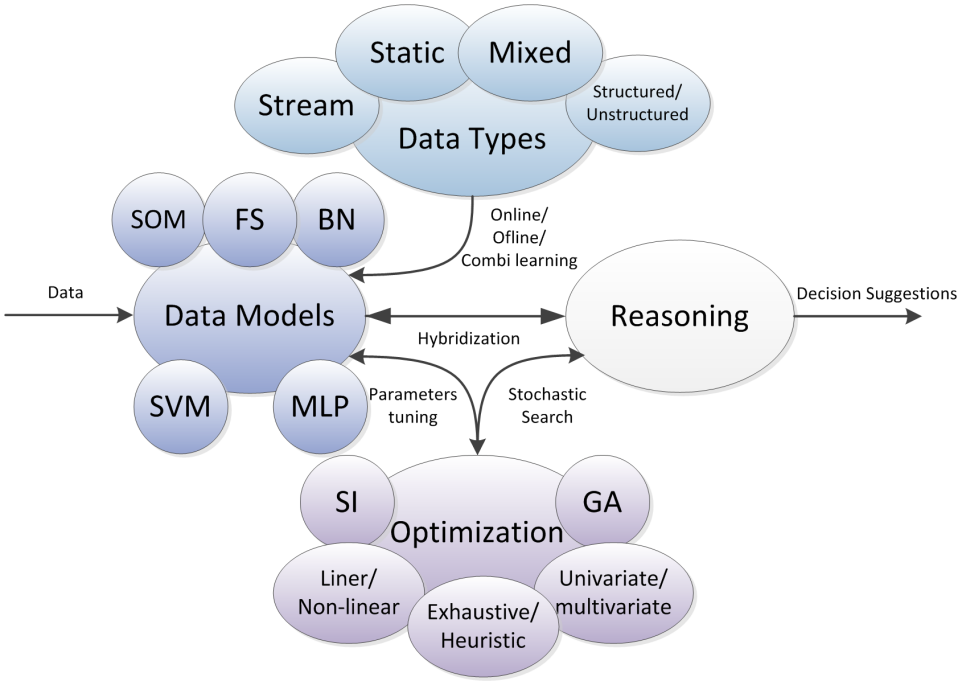


Figure 2.16: Hybridization of SC with respect to different factors

when dealing with large-scale datasets. The hybridization improves quality and understandability, yet increases arithmetical complexity of the ensembled method. The last challenge can be mitigated by means of parallel meta-heuristic techniques as Yang studied in 2010 in the book [448]. We can say that there already exist several solutions for storing and mining the large-scale data such as Mahout, ML-Base[32] and Spark[33]. However, they are targeted only on commercial applications such as customer recommendation systems, etc. As a result, we are sure that hybridization together with parallel meta-heuristic optimization can handle multiple challenges in Forensics Sciences.

Forensics Science and Big Data. The amount of information analysed in criminal cases grows each year, especially related to computed incidents. Since conventional SC methods originally are not optimized for such large-scale data, we consider hybridization as a prominent solution for this problem. Moreover, meta-heuristic optimization is considered to be suitable for SC.

Information Fusion. Information Fusion represents a general methodology used

[32] http://www.mlbase.org/
[33] http://spark.incubator.apache.org/

to merge and combine existing data and parameters to achieve better results of more accurate models. Torra [416] stated three possible paradigms that can be used in Information Fusion such that: pre-processing, model building, and information extraction. Our particular interest is model construction that is a combination of several models to achieve final module simultaneously or over time. This is an inseparable part of Information Security community considering a number of sensors that can retrieve information from any computer system. There has been a variety of work that targets this area recently. Bist et al. [81] investigated how feature fusion can be applied to computer virus detection and study its influence on the application of pattern matching methods. Similar work was performed by Modi et al. [287] with respect to Threats Intelligence, where they proposed a novel framework for supplementing malware analysis with different threat sources. Another prominent application of Information Fusion is Intrusion Detection. Raja et al. [330] described the collaboration of IDS system with the application of sensors data fusion that showed good performance on the KDD dataset. Another work on sensors information fusion was done by Pugh et al. [324]. Thus, there are examples of many applications and we believe that it is quite important to apply Information Fusion with respect to growth in the amount of data and possible attack vectors. There has not however been done much work on Information Fusion for malware Digital Forensics, especially using Fuzzy Logic. Despite its applicability, the NF technique possess a weakness that affects its scalability during alteration in the fuzzy set L_i. There exist several techniques to deal with the changes in the NF model. First, the *off-line* re-training of the model is done from scratch [155]. This causes significant delays and infeasibility when the duration of re-training is greater than the time between the term's set changes. Second, the *on-line* incremental learning over a fixed set of linguistic variables can be utilized, for example ANFN [236] or self-organizing neuro-fuzzy OSNFS [430]. In this case, the Neural Network is learning from the data streams. This however requires re-training of the model to keep the changes in characteristics. Therefore, the study proposed splits and merges in rules sets in the study [257][Chapter 5] to mitigate the challenges.

2.5.2 Required Hybridization & Kosko Model

Considering the various weaknesses of each of the methods, different types of hybridization can be used in order to support and optimize the discussed SC methods. To counter this, each method can be supplemented in a very specific manner in order to comply with the "no free lunch theorem" introduced by Wolpert et al. [442].

Despite possible improvements, the implications of the wrong computational results like non-deterministic, non-optimal solutions may appear. Another challenge is that some of the aforementioned problems can be optimized only to some levels of complexity. Due to the inability of making them simpler, HI mitigates the chal-

lenge by providing an inexact solution and using parallel meta-heuristics. As a result, the problems solved in a faster manner using parallel computing. In this work, we look into the Neuro-Fuzzy system initially used by Kosko [233] shown in the Figure 2.17.

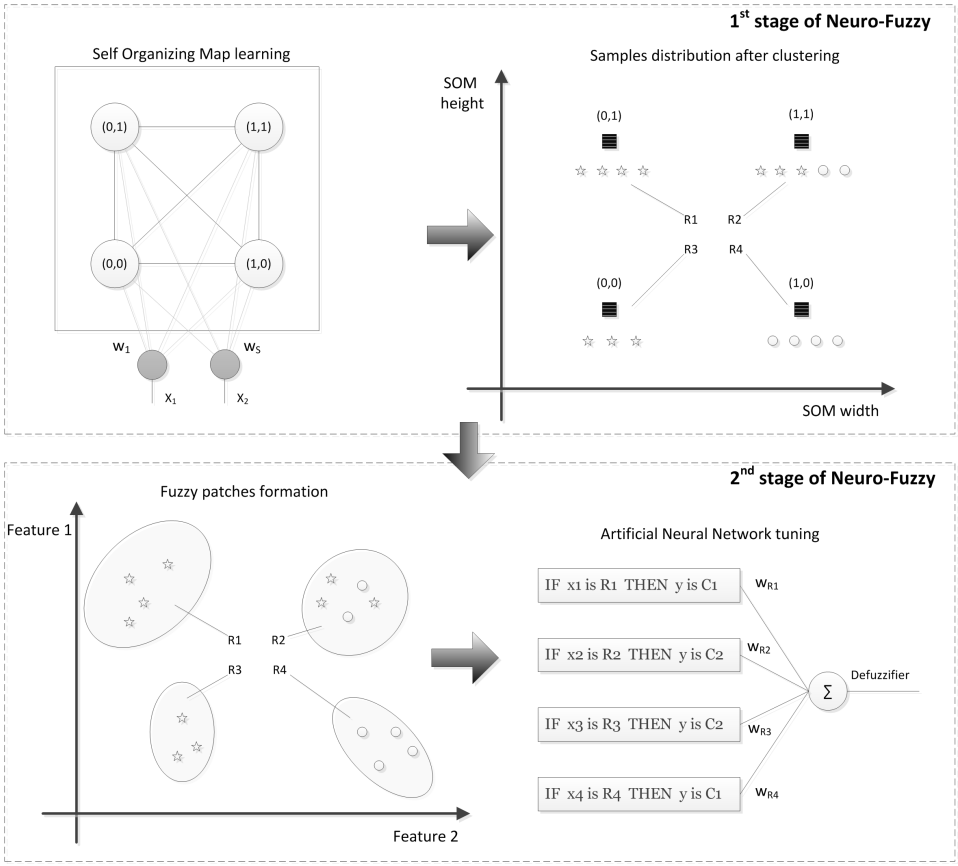


Figure 2.17: Neuro-Fuzzy approach that includes two stages [233]

Neuro-Fuzzy is a Hybrid Intelligence method that assembles FL and ANN into a classification model. The major data operations of NF are divided into two logical stages [233]:

1^{st} **NF stage** is an unsupervised procedure that is aimed at grouping samples according to their similarity. The input data sample is a real-valued vector $X = \{x_i \in R, \leq i \leq M-1\}$, as in a corresponding set of features $X = \{x_0, \ldots, x_{M-1}\}$, and it characterizes a point in M-dimensional space (number of features in input data). SOM is trained, resulting in groups of samples to be later used for fuzzy

patch construction. As can be seen, the main challenge is the determination of the number of SOM nodes, e.g. width and height of the map. The peculiarities of existing approaches for SOM training will be described in a later section. The result of this stage is a set of clusters that each form a fuzzy patch based on the statistical parameters inherent in each group. These patches are either rectangular or elliptic. 2^{nd} **NF stage** is a supervised procedure. On this stage, a set of fuzzy rules (based on fuzzy patches) are fed to ANN with their corresponding weights assigned. The iterative training procedure results in an accurate classification model employing the discussed fuzzy rules.

2.5.3 Self-Organizing Map Configuration[34]

Self-Organizing Map (SOM) or Self-Organizing Feature Map (SOFM) or Kohonen map is one of the most powerful Neural Network-based unsupervised models. Among various applications, it is used for 2D data representations and data grouping. The main difference from clustering is that the data samples are grouped according to their similarity, not just by distance between a cluster center and any of the samples. This model was originally proposed by Teuvo Kohonen [229], where he described self-organizing systems as a one- or two-dimensional matrix that has a feedback connection between neighbouring nodes in the map. Most of the presented interactions were based on one-dimensional output representation of the input. The resulting formation depends on the feedback level and order of the samples being fed into the map making, possibly automated formation of the similarities from map topology.

In this subsection, we focus on the peculiarities of SOM and what is their influence on the Neuro-Fuzzy model. The main challenge is the results SOM clustering. As per today, this unsupervised procedure mostly follows the structure of the grid defined by the analyst. Alternatively, the SOM size changes iteratively over learning like it is done in Growing SOM [50]. This is an important issue in Digital Forensics application since this has to be done without manual support. Moreover, the size of SOM influence the fuzzy rules construction that needs to be understandable and easy interpretable that will be shown later

SOM can be organized as *fixed-size* or *growing* topology according to Valova in the "SOMs for Machine Learning" [459, Chater 2] in order to find an optimal size, that better fits presented data. We will concentrate our attention on the fixed-size SOM since it does not require additional knowledge about heuristics in data [186]. Also, the amount of computational resources is lower when the fixed-size rectangular SOM is used. There exist several challenges to finding an optimal size since this has a direct influence on the set of the fuzzy rules. Each rule is a characterisation

[34]The main ideas of this subsection are published under the contribution [374, 376]

of similar data samples derived by means of SOM. Too general clusters will cause underfitting, while too specific clusters will cause underfitting of the model [93]. From the other side, well-packed clusters will give more empty clusters when the size of SOM is big. The author in [137] stated that the maximum grid size should be equal to $5 \cdot \sqrt{N}$, where N is the size of the dataset. This can be defined as a "rule of thumb" is SOM size definition. With such a metric, the number of nodes converges to infinity when dealing with Big Data. The data modification is not favourable due to the loss of original properties through the addition of abstraction levels or transformation [394]. Furthermore, a smaller set of rules is more appropriate for data representation in a Court of Law.

Since the most commonly used empirical measure $5 \cdot \sqrt{N}$ is SOM toolbox[35] according to Vesanto [423], there is a need for more precise definition of the number of nodes. At this point there are several options how the proper size can be estimated. The Growing SOM [459, Chater 2] learns from data iteratively, adding new nodes. This process might however give too big a topology of SOM resulting in overfitting [93]. Moreover, the SOM can be run several times to estimate the average number of clusters after learning as a simple approach. Since the SOM learning process is based on a random component, it gives similar results that slightly differ from the optimal size. Finally, we in the research [180], the biggest eigenvalues of the dataset were used to tune these parameters. The eigenvalues ratio shows how well the data flattened and elongated [137]. In other words, the SOM grid needs to be spanned respectively. This is used in Factor Analysis to determine the proper number of factors to be used with respect to the covariance fraction, though it does incorporate the information about structure and dependency in the data. Thus, estimation of optimal SOM size is done mostly empirically. The optimal heuristic boundary of empty nodes is 5-10% after the SOM is learned.

Fundamentals

The training principle of SOM follows standard routine of ANN training, making some additional abstract layers required in mapping from multidimensional data to Cartesian coordinate systems containing two pseudo-axes SOM_1, and SOM_2 for simpler representation. One can use different colouring schemes for visualization to show various clusters in the data. Generally speaking, SOM network has a format that is depicted in the Figure 2.18. SOM also reduces the dimensionality of input data only two dimensions. SOM consists of so-called *nodes* (denoted as a circle), where each node is bound with all features in the input vector by means of weighted connections, as was described in early work by Kohonen [229].

SOM training is done through assigning each input sample to its most similar node

[35] http://www.cis.hut.fi/somtoolbox/

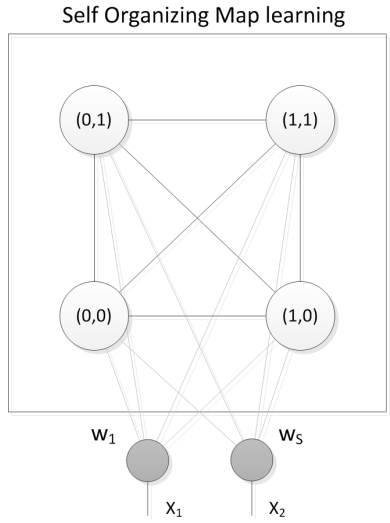


Figure 2.18: A general concept of Self-Organizing Map architecture

on the grid [115]. The goal is to have a stack/vector of similar input data samples grouped in a corresponding node. Say we have multiple input data samples or a format $X_i = \{A \in R^M\}$ with a corresponding set of features $A = \{x_0, \ldots, a_{M-1}\}$, and it can be characterized as a point in M-dimensional space. Each node (m, k) on the SOM network will have a set of M corresponding weights $W_{mk} = \{w_0, \ldots, w_{M-1}\}$ connected to the input of the network. The process of training (weights adjustment and input data assignment) can be described by the following [163]:

1. *Weights initialization* is done by assigning a small random value to each of the weights in the model.

2. *Training* process consists of several important steps. At this point, input data samples X are fed into the network randomly or based on some predefined order.

   (a) Calculation of Best Matching Unit (BMU) based on the Euclidean distance between the input data X_i and a node weights vector W_{mk}. Input data will be assigned to a node with the smallest distance:

   $$D_{BMU} = ||X_i - W_{mk}|| \tag{2.5}$$

   (b) *Updating neighbourhood* means that the influence of the node on other nodes will be reduced based on the exponential decay function $h(t)$

over time (number of iterations) t. This region of influence will be reduced over time according to [115]:

$$h(t) = \sigma \cdot e^{-\frac{N_t}{\lambda}} \tag{2.6}$$

where σ is a radius of the initial neighbourhood around each node, considering it to be a circle with a size no more than the size of the SOM grid:

$$\sigma = \frac{max\{SOM_{width}, SOM_{height}\}}{2} \tag{2.7}$$

λ is a speed or a time constant that indicates how fast the region will be reducing over time or a defined number of iterations N_{som}, and a current iteration N_t. λ is calculated on each iteration according to the following equation:

$$\lambda = \frac{N_{som}}{log(\sigma)} \tag{2.8}$$

(c) *Learning rate adjustment* is performed using constant initial rate L_0, as normally done in the general ANN training:

$$L(t) = L_0 \cdot e^{-\frac{N_t}{\lambda}} \tag{2.9}$$

(d) *Weights update* is performed with the aforementioned parameters for each node (m, k) in SOM:

$$W_{mk}^t = W_{mk}^{t-1} + L(t) \cdot \theta(t) \cdot (X_i - W_{mk}^t) \tag{2.10}$$

where θ denotes the effect of the distance from a node to BMU:

$$\theta(t) = e^{-\frac{D_{BMU}}{2 \cdot h^2(t)}} \tag{2.11}$$

3. *Final data representation* is the extraction of the samples grouped around each node in SOM, meaning that they are similar.

SOM becomes integrated in the NF model on the 1^{st} stage described by Kosko [233], since the main task is to learn the relation and properties of data in order to be able to build fuzzy rules automatically on the 2^{nd} stage.

Interpretability Concerns

The biggest challenge in building a fuzzy rules model lies in finding an optimal trade-off between accuracy and interpretability. In the research, [199], Ishibuchi stated it as a multi-objective problem as follows for the fuzzy-based system F and size of the SOM grid S (number of nodes):

$$\max_{S} \left(Accuracy(F),\ Interpretability(F)\right) \tag{2.12}$$

It is not always easy for forensics analysts to define the appropriate parameters, especially when dealing with Big Data. Thus, we consider that the fuzzy-based method in the range from 2x2 to 5x5 can be reasonable based on the complexity of the model. Moreover, any kind of optimization can be used in the Equation 2.12. There are many obstacles, and no unique solutions are available for the optimal size of the fuzzy-based systems. A more extensive study of interoperability measures was done by Gacto et al. [156]. The authors highlighted the need for a trade-off between accuracy and interpretability. The accuracy can be measured easily, while the interpretability measures are not easy to define. Some of the main features used to study their interpretability were the number of rules, number of MF, etc. The number of conditions in the rules should not exceed 7 ± 2, which is a boundary that the human brain can handle. The number of features can sometimes however be much more than this limit. Therefore, the only way to limit the complexity of the model is to shorten the number of rules. Another work [183] on the trade-off between accuracy and interpretability is done by Herrera et al. The authors tried to lower the number of rules by means of a Taylor series approximation. The authors showed that usage of up to 3 MF in each dimension can be considered as sufficient for a better accuracy on a dataset. The drop of accuracy in the fuzzy systems with higher number of rules is caused by a spread and over-fitting of the training data. According to Alonso [53], the maximum number of rules acceptable by user should be greater than or equal to 10^3 times the number of classes. Castellano [89] presented A Priori Pruning for the human-understandable rules selection from NF. Only 66 rules were extracted from more than 200 thousand of the rules produced by the grid partition. So, there is a need to bound the grid size to improve the generalization, and it is reasonable to use less than 5^2 rules for the human-understandable model.

Estimation of the SOM Size

In this work, we consider the rectangular topology of SOM that gives rectangular mapping of the region, which is more appropriate for the fuzzy terms extraction. The reason for this is that it provides more abrupt clusters, grouping the samples within edges rather than overlapping with better connectivity as was studied by

Baltimore [73]. Below, we give an insight into existing methods and schemes. To the authors awareness, there is not yet a method for find an optimal size of SOM that complies with the Daubert Standards for the models to be tested and interpreted according to the stated requirements [146]. The main problem is in presenting fuzzy rules from SOM clustering without altering the original data in a forensically-sound manner. Most of the existing methods generate a big number of specific rules.

Modification of topology

One of the possible techniques to finding an optimal size of SOM (number of nodes) is the iterative process of topology modification. In most methods, this process starts with a small number of nodes, while constantly adding new nodes until some quality stopping criteria is fulfilled. One of the first modification of Kohonen SOM was described by Alahakoon et al. in 1998 in the research [50]. In this work, the problem of optimal size definition was first addressed by introducing the Growing SOM (GSOM). Authors proposed the use of additional growing phases that add new nodes with specific coordinates until the growing conditions are no longer satisfied. New nodes are added to the boundary while the entire structure is preserved. Then, the weights of this node are given from the bounded values. The method has a good speed on the small dataset since the initial number of nodes in the topology is only 4. Further analysis of this method for knowledge discovery was performed in 2000 through the introduction of the Growing Hierarchical SOM (GHSOM) [51]. In the paper, it was stated that the general guideline is to generate a smaller map first and then expand the analysis process in each cluster. Yet it is necessary to define the growth threshold GT and the spread factor SF. This requires additional knowledge and work done by the data analyst. Thus, this method requires specific knowledge of the dataset that the malware analyst may not have. Another challenge is that the fuzzy relationships cannot be represented properly since the boundaries on the different layers are crisp. To deal with this challenge, several other methods were proposed for fuzzy rules derivation while using SOM in the modification of the topology. The first one, Adaptive Self-Organizing Neural Network (GSFNN) was proposed by Qiao in 2007 [145]. This method requires an intense computation process together with a bunch of additional parameters that need to be defined empirically by an additional data analytic. Our concern is with the complexity of the methods mentioned above; since growth is not bound, the structure may be too complex. As a result, this would take too much time to tune the parameters on the 2^{nd} stage of the NF method. At this point, we will consider fixed-size models to be more suitable for use in NF methods than changing the topology, since then we will not be able to control the granularity of the fuzzy sets.

Based on the dataset size

Another approach can be used when the SOM topology is fixed and the size of the map is fixed too. It was mentioned that the number of nodes should be no more than the number of samples in dataset. Thus, a non-empty SOM node should consist of at least one sample. In [137, 423, 358], the applicability of the quantity metric defined by Vesanto in 2000 for the SOM Toolbox in Mathlab [424] suggested that the optimal size of the SOM is empirically determinable. The next definition of the grid size refers to the dataset size.

$$S = 5 \cdot \sqrt{N} \tag{2.13}$$

Definition 1: The initial size of the SOM is equal to $S = 5 \cdot \sqrt{N}$. The boundaries of the size are defined as follows, according to SOM Toolbox:

$$S_{lower} = 0.25 \cdot S = 2.5 \cdot N^{0.54321} \tag{2.14}$$

$$S_{upper} = 4 \cdot S = 20 \cdot N^{0.54321} \tag{2.15}$$

Furthermore, the number of nodes in each of 2 dimensions of the rectangular SOM are calculated as following mapping to integer numbers:

$$S_a = \left\lceil \sqrt{S} \right\rceil, S_b = \left\lfloor \frac{S}{S_a} \right\rfloor \tag{2.16}$$

This is according to SOM toolbox documentation [424], and was measured empirically. We can see the following problems with this measure: initially, when the number of data samples is huge, the size of SOM will grow infinitely:

$$\lim_{N \to \infty} S = \infty \tag{2.17}$$

Thus, this SOM likely would give most of the nodes in the empty state. Finally, with the huge number of derived regions we cannot make a simple and understandable classification model.

Using variance

The last possible way of determining the best SOM size is the analysis of the spread of the data, for example the measure of variance. Principle Component Analysis extracts the components that can be characterised by the distribution of variance [396]. It means that by means of eigenvalues for each corresponding eigenvector, the characteristics of the distribution can be measured. This technique is used in Factor Analysis [47] to see how well the factors fit the model and whether they influence it at all. As defined in the Equation 2.13, the size of the SOM may

be changed in case of variance on the first components with the biggest eigenvalues take the majority of the variance in the dataset [137, 423].

Definition 2: The distribution of the data along the principle components characterise the way that most correlated components are distributed in the data. By applying the following formula:

$$S^{'} = 5 \cdot \sqrt{N} \cdot \frac{e_1}{e_2} \tag{2.18}$$

where e_1 and e_2 are two eigenvalues with the biggest values from the work by Vesanto [423]. This can be interpreted as following data spanning. If both vectors are nearly equal, then the more general cluster will be composed by the number of nodes S. Otherwise, more nodes will be used in $S^{'}$ if the data are stretched along one of the components, which implies significant correlation.

Role of correlation

One of the main factors that influences the goodness of fit of the fuzzy model to the data is the correlation between the features [209]. If there is a correlation between the features present in the data, then a larger amount of fuzzy patches is required to cover the area. However, this amount is still lower than in the case when all possible combinations of MFs are constructed. The basic linear dependencies can be easily revealed by means of the widely-used Pearson Correlation Coefficient (PCC) r. It is applied when fast and tentative information about the relations in the dataset is necessary. The PCC between two variables is calculated as is shown in the Equation 2.19.

$$r = \frac{Cov(XY)}{\sigma_X, \sigma_Y} \tag{2.19}$$

where σ denotes the standard deviation for each attribute, and $Cov(X, Y)$ the covariance between two attributes. The complexity can be defined as follows. It requires $1 \cdot N$ computations of the mean $\bar{x}$, N computations for the covariance between them, and $2 \cdot N$ computations for the variance. Overall computational complexity is $O(M \cdot N)$ on the M-dimensional dataset with N samples. Additionally, this does not require the storage of any supplementary matrices.

The Big Data that needs to be analysed in Digital Forensics has complex dependencies [169]. Different correlation techniques can be employed to incorporate the outlined challenges, including confounding variables and non-monotonic dependencies. According to the study [103], there are several metrics of the correlation such as PCC, Spearman, Distance Correlation (DC), and Maximal Information Coefficient (MIC). The DC is a new measure that was proposed in [407] by Szekely

et al. in 2007, and is based on the Euclidean distance covariance between the variables. The Spearman and PCC correlations are defined as linear. Spearman does not differ much from PCC, though more iterations are required to find and store ranking meta-data. On the other hand, DC and MIC have been defined as non-linear metrics. PCC, DC, and MIC metrics were successfully used in the associations discovery in large astrophysical databases by Martinez-Gomez et al. [267]. According to the work, DC is more effective than MIC and better reveals associations in the datasets.

$$dCorr = \frac{dCov(X,Y)}{\sqrt{dVar_X \cdot dVar_Y}} \tag{2.20}$$

The distance covariance $dCov(X,Y)$ and variance $dVar_X^2$ are defined as the following measure of the element-wise Euclidean distance matrix $A(a_{ij} = ||X_i - X_j||)$ and $B(b_{ij} = ||Y_i - Y_j||)$ [407]:

$$dCov(X,Y)^2 = \frac{1}{N^2}\sum_{i,j=1}^{N} A_{ij}B_{ij},\ dVar_X^2 = \frac{1}{N^2}\sum_{i,j=1}^{N} A_{ij}^2 \tag{2.21}$$

Considering the complexity, this method needs N^2 Euclidean distance computations between every sample's attribute value, N^2 computations for covariance and $2 \cdot N^2$ computations for variance. Overall, the complexity can be defined as $O(M \cdot N^2)$ for the input data. Also, it requires $2 \cdot N^2$ of memory structures required to store distance matrices, A and B on every iteration. Despite the fact that DC is a non-linear measure, it was shown in [267] that there is also a sharp dependency between it and the PCC. It makes DC overhead $O(M \cdot N) < O(M \cdot N^2)$ less reasonable when employed in Big Data analytics in comparison to the PCC results.

Thus, PCC and DC are the most promising measures used in the determination of optimal SOM size. Even considering that PCC was defined as a linear one, it has a strong compliance with the defined non-linear DC. At this point, we can also state that DC is less efficient on huge datasets due to computational complexity. Moreover, DC takes more space for meta data when $N >> M$ than for the dataset itself. In this light, it is more efficient to use PCC for the estimation of the dependencies in the dataset.

2.5.4 Fuzzy Patches Revisited[36]

So-called fuzzy patches Π_{XY} define the relationship between an input fuzzy set X and output fuzzy set Y. As a result, the outcome of the SOM clustering can

[36]The main ideas of this subsection are published under the contributions [373, 376]

be represented by a set of clusters or groups of similar instances. Each cluster or patch will result in a corresponding fuzzy rule with a particular class label, since these are Mamdani-type rules. Therefore, an arbitrary fuzzy patch Π_{ij} represents an input-output mapping of the input feature vector $X_i = \{A \in R^{N_F}\}$ with feature set $a = \{a_0, \cdots, a_{N_F}\}$, and output class vector Y_i. Below, we discuss the different most commonly used types of fuzzy patches and how they are constructed from SOM clusters according to Kosko [233].

Definition 3: *The simple rectangular patch* represents a N_F-dimensional hyper-rectangle (N_F features in the dataset) that contains inscribed instances of a particular cluster as shown in the Figure 2.19. The boundaries of the patch are derived using 1_{st} and n^{th} order statistics for each of the features a_i:

$$B_{min}^{a_k} = min\{\Pi_i^k, \cdots, \Pi_i^k\} \tag{2.22}$$

$$B_{max}^{a_k} = max\{\Pi_i^k, \cdots, \Pi_i^k\} \tag{2.23}$$

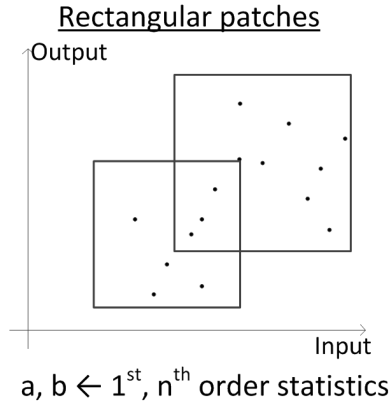


Figure 2.19: A simple fuzzy patch which defines an arbitrary rectangular region

Considering that the patch is a simple rectangular, we can define the center and the length of each side that corresponds to a particular feature a_k space:

$$L^{a_k} = B_{max}^{a_k} - B_{min}^{a_k} \tag{2.24}$$

$$C^{a_k} = \frac{B_{max}^{a_k} + B_{min}^{a_k}}{2} \tag{2.25}$$

Definition 4: *The elliptic patch* in the Figure 2.20 used by Kosko back in 1997 [233] has a M-dimensional hyperellipsoid form that is circumscribed around the data in a cluster:

$$(x - c)^T (x - c) = \alpha^2 \tag{2.26}$$

where α is a pseudo-radius of the fuzzy patch for orthogonal uncorrelated features, and c_i is a centroid of a particular feature in the cluster. It can be explained as a set of N data samples, and is therefore contained in an N_F-dimensional ellipsoid (hyperelipsoid) with a radius α, of general form $\sum_{n=0}^{N_F-1} x_i^2 = \alpha^2$. This is the shape of the multidimensional spread of data around some central tendencies caused by Gaussian distribution. This is the distribution that defines most real-world processes.

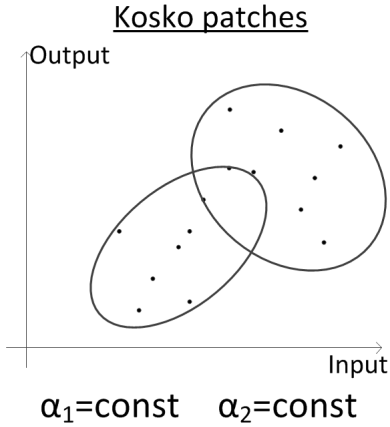


Figure 2.20: Ellipsoid fuzzy patches used by Kosko [233]

Remark 1. It was assumed that the data in the cluster was distributed so that a fuzzy patch Π_{ij} forms a multivariate distribution. Such fitting will reduce error of the data from outside the patch and provide better data-fitting, as stated by Bertoni [78]. The difference between simple rectangular and elliptic patches is shown in the Figure 2.21. As can be seen, there is a much higher degree of error when using rectangular patches.

However, this equation does not incorporate the information about any correlation in the data that will result in the rotation and stretching of the hyperellipsoid. Therefore, an inverse covariance matrix Σ^{-1} will be incorporated.

$$(x-c)^T \; P \; \Lambda \; P^T \; (x-c) = \alpha^2 \tag{2.27}$$

At this point $\Sigma^{-1} = P \; \Lambda \; P^T$ represents a positive definite symmetric matrix, which is factorized using eigendecomposition as described by Abdi [36] into the diagonal matrix of eigenvalues $\Lambda = (\lambda_1, \ldots, \lambda_{N_F})$ and orthogonal matrix of eigenvectors $P = (e_1, \ldots, e_{N_F})$ that rotates the ellipsoid. It should be noted that according to the Eigen decomposition theorem described by Abdi [36], the covari-

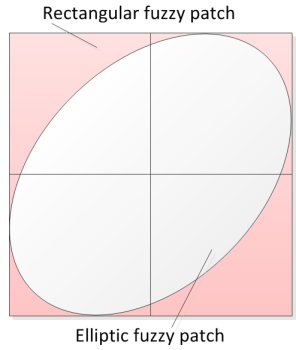


Figure 2.21: Differences in data coverage provided by simple rectangular and elliptic fuzzy patches

ance matrix is transformed into decomposition $\Sigma^{-1} = P\ \Lambda\ P^T$ since eigenvectors are orthogonal.

Remark 2. We can then refer to Principle Component Analysis (PCA) presented by Smith [396]. On the initial step of PCA, the eigendecomposition is performed to find out the components with highest degrees of variance. Then, the initial eigenvalues from the ordered vector represents the components with the highest lengthiness along the eigenvectors. This information can be used to find out whether the amount of required SOM nodes should be increased to cover the stretched region, as was mentioned earlier. Furthermore, the radius of each corresponding ellipsoid axis is equal to $\alpha/\sqrt{\lambda_i}$ due to $\lambda_i \cdot (x_i - c_i)^2 = \alpha^2$ according to Kosko [233], since for the orthogonal matrix $P^T = P^{-1}$, which would eliminate the eigenvectors values in Equation 2.27. As can be seen, the definition of α^2 in Equation 2.27 determines the efficiency of the method. So far, this parameter has been defined empirically from data.

To summarize, clustering is done by trained SOM based on the features similarities to convert an M-dimensional feature vector into a $2D$-lattice that consists of $H \times W$ nodes. After training each node, $S_{i,j}$ includes cluster of similar data samples as depicted in the Figure 2.22. One can see that a single SOM node can contain samples belonging to one or more clusters that characterize a particular class (samples of A or B classes in the Figure).

Figure 2.22: Extraction of elliptic fuzzy patches

The construction of elliptic regions is done to improve the Kosko method. We make a hypothesis that the multivariate distribution defines the model [383] for the data clustered in the node $S_{i,j}$ in SOM (referring to the Figure of histograms from different features from the dataset):

$$g(x) = \frac{1}{\sqrt{(2 \cdot \pi)^M |\sum|}} \cdot e^{-\frac{1}{2}(x-\bar{x})^T \sum (x-\bar{x})} \tag{2.28}$$

Based on the this assumption and the properties of Gaussian multivariate distribution, there need to be defined corresponding parameters $\{\bar{x'}, \bar{\sigma'}, \sum\}$ for the given data. The distribution model represents an n-dimensional elliptic region or fuzzy patch for the cluster in Euclidean geometry.

2.5.5 Membership Functions Basics[37]

MF in a fuzzy set usually takes the values $[0; 1]$ and is used to define the degree of truth to which the samples belong to a particular rule. Since the rule is composed from many transformed feature linguistic terms, we need to define the MF for each of the feature spaces. There are two definable membership functions from before that were used in Neuro-Fuzzy, according to Kosko [233]:

Definition 5: The *triangular MF* is used together with rectangular fuzzy patches and has the following form:

$$\mu_j(X) = \begin{cases} 1 - \dfrac{|x_j - c_j|}{p_j}, & |x - c_j| \leq \dfrac{p_j}{2} \\ 0, & otherwise \end{cases} \tag{2.29}$$

where c_j defines a center of the triangle for the i-th feature and p_j is the corresponding length of the base of the triangle.

Remark 3. The triangular MF is derived from the rectangular fuzzy patches and has the following parameters used in the Equation 2.29:

$$c_j = C^{a_j};\ p_j = L^{a_j} \tag{2.30}$$

Triangular MF is the generally-used type of function and its parameters are easy to derive from data, as shown in the Figure 2.23. Therefore, we included this MF for comparison in our work. According to Dickerson et al. [120], triangular MF was used to simplify the derivation of rules. This method uses 1^{st} and n^{th} order statistics in a cluster, assuming it fits the rectangular region and then the simple triangular MF to characterise each region.

[37] The main ideas of this subsection are published under the contributions [373, 376]

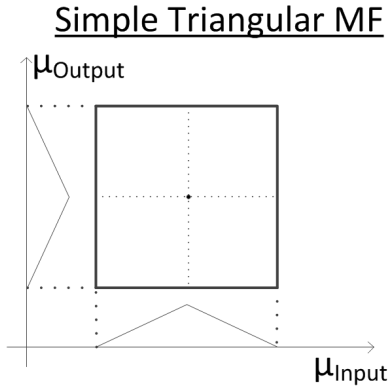


Figure 2.23: Simple Membership Function used to defined the degree of truth in rectangular fuzzy patches [233]

However, when it comes to elliptic fuzzy patches, MF construction is not that simple. The input data sample is $X_i = \{A \in R^M\}$ with the corresponding set of features $a = \{x_0, \ldots, a_M\}$, and can be characterized as a point in M-dimensional space. The whole set of N data samples is therefore contained in an M-dimensional ellipsoid (hyperelipsoid) with a radius α and center c_i. Furthermore, to include the correlation between the features, we need to include the covariance matrix that also defines the rotation of the elliptic region in the Kosko method:

$$(x - c)^T \, \Sigma^{-1} \, (x - c) = \alpha^2 \tag{2.31}$$

where Σ^{-1} is a definite positive inverted symmetric covariance matrix. The number α is set to be the same for every fuzzy patch as in the Kosko method.

The Kosko method employs elliptic modelling instead of the previously mentioned rectangular patches, and then derives the corresponding parameters of the triangular MF as presented in the Figure 2.24.

To simplify the MF construction, Dickerson and Kosko [120] defined a rotated rectangular region, in which the hyperelipsoid is inscripted. As a result, the following triangular-based MF is constructed:

$$\mu_j(X) = \begin{cases} 1 - \dfrac{|x_j - c_j|}{p_j}, & |x - c_j| \leq \dfrac{p_j}{2} \\ 0, & otherwise \end{cases} \tag{2.32}$$

where the μ^j defines the MF of j feature and the projection of the circumscribed hypperrectangular on the i axis

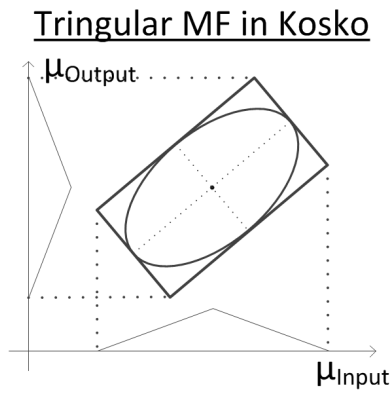


Figure 2.24: Representation of the membership function together with elliptic fuzzy patches

Definition 6: Projection-based triangular MF according to Kosko [233], the MF in Equation 2.32 is constructed using the projection from a hyperellipsoid on each feature axis. The base length of the triangle for the j-th feature space is defined as a sum of the projections on the i-th axis:

$$p_{ij} = 2 \cdot \alpha \cdot \sum \frac{|cos\theta_{ij}|}{\sqrt{\lambda_i}} \tag{2.33}$$

where angle between the i-th axis and j-th eigenvector e_j:

$$\theta_{ij} = arccos(e_j(i)) \tag{2.34}$$

Figure 2.25 shows the resulting ellipsoid projections that were described by Kosko [233].

Since the unit eigenvector represents the vector of direction cosines between the principle axis of rotation and original features axis as it is eliminated from the characteristic polynomial $A \cdot e - \lambda \cdot e = 0$. In Euclidean space, the features' vectors are mutually orthogonal as well as the corresponding set of eigenvectors. Therefore, it is feasible to use a set of eigenvectors as a rotation matrix in eigendecomposition of the inverse covariance matrix Σ^{-1}. The projection of the hyperrectangle circumscribed around the target hyperellipsoid defines the parameters of this MF. We can see that such a projection does not fit the data properly since the data might contain outliers. In the above mentioned research, the authors stated that this way of MF composition does not use the orientation of the ellipsoid, and might be improved by utilization of the ellipsoid patch itself in MF. The following Section gives a

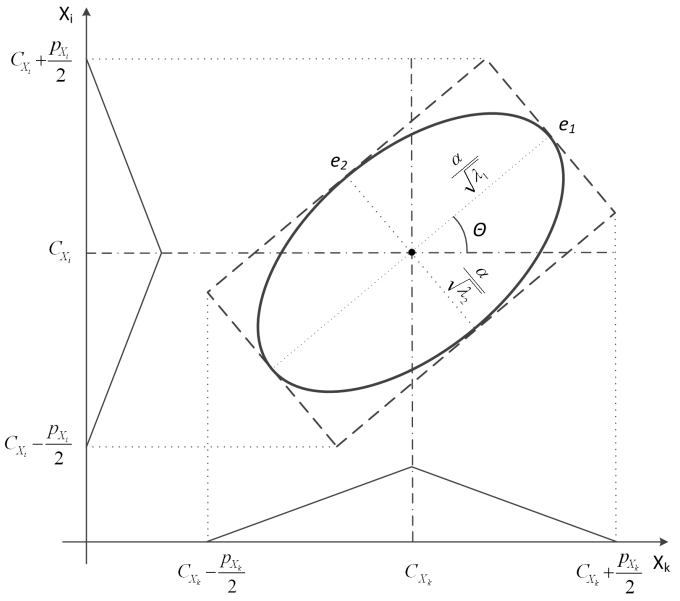


Figure 2.25: Projection of the eliptic fuzzy patches on the axis according to Kosko [233]

view on the probabilistic modelling in the definition of parameter α^2. Moreover, the modified MF function is presented based on the multivariate distribution.

2.5.6 Tuning of Fuzzy Rules

Artificial Neural Networks play an important role in NF architecture not only as an unsupervised method for fuzzy patch allocation, but also for the fine-tuning of extracted fuzzy rules. Different NF architectures such as ANFIS described by Abraham [39] use Neural Network learning for enforcing the accuracy of the Fuzzy Inference model, and adjusting the corresponding weights in the Neural Network, whose role is of supervised trainer on the rule inference layer. By applying such training, the NF model gets a better-expressed rules firing process, where rules are considered to be input to ANN. A general scheme of ANN is shown in the Figure 2.26.

As written before, ANN is known to be one of the most powerful ML methods capable of learning from erroneous, complex, and incomplete data. Generally, its training is done via minimization of the objective function of the error signals $E(W)$:

$$E(W) = \frac{1}{2}(y - d)^2 \tag{2.35}$$

where d - desired output of the ANN, y - actual output, and W is a set of all

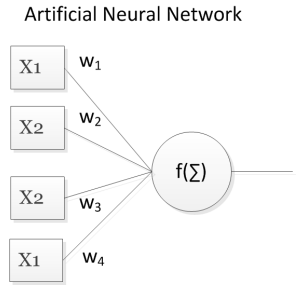


Figure 2.26: A general representation of Artificial Neural Network [232]

weights. The main obstacle in learning is that the method can be stuck in a local optima unless the learning rate is an optimal one. Thus, the primary optimization problem in the ANN is the minimisation of the function:

$$min\ E(W)\ |_{W \in \mathbb{R}^M} \tag{2.36}$$

where each function in the $E(W)$ is an objective function of the neuron's weights w_i^j ($j-th$ hidden layer and $i-th$ hidden unit) that should be optimized with the following condition on the whole domain of the function $dom\ E(W) = \mathbb{R}$ and with respect to the learning rates α:

$$\forall\ w_i^j\ \in\ \mathbb{R}:\ E(w_i^{j^*})\ <\ E(w_i^j) \tag{2.37}$$

Researchers have proposed different learning schemes including complex algorithms like Widrow-Hoff LMS and Adaline as described by Widrow et al. [438]. The MLP learning process should be optimized as a differentiable error function, and then the Gradient Descent (GD) optimization of the function tunes the weight of the neurons:

$$w_i^{j^{new}} = w_i^{j^{current}} - \alpha \cdot \nabla E(w_i^j) \tag{2.38}$$

where $E(w_i^j)$ is a multidimensional error function over a weight w_i^j. The principle of the learning then is to use the so-called Delta Learning rule, comparing the output of the network against the labelled dataset. The Delta Learning Rule makes a robust first-order approximation as stated by McClelland et al. [273] along the partial derivative direction only if the learning rate is less than or equal to the optimal one, also described by Mandic et al. [262]. In real-world tasks however, it is hard to predict how the optimal learning rate will change under the influence of an input concept drift. The determination of the learning rate α brings with it the most challenge. There are several options for definition, such as constant rate or iterative adjustment [232].

2.5.7 Binomial & Multinomial Classification[38]

Mamdani-type NF is a model which is specifically designed for classification problems, where each fuzzy rule denotes a specific group of samples that can be denoted with a fixed label, as studied by Kosko and Chen [95, 233]. The advantage of fuzzy rules is that they can describe each group of similar data that belong to different classes separately without designing a separation hyperplane as is normally done in other binary classification methods. Considering also the area of Digital Forensics, such a model underwrites great interest in building human-understandable and interpretable models that are also accurate. The majority of the tasks in illegal activity detection denotes either *"malicious"* or *"benign"* patterns, which is a binary classification task. These can be software samples, network traffic dumps, web pages, etc. There are studies however that require the determination of a specific group or domain which this *"malicious"* pattern belongs to, for example network attacks or malware families.

Many methods such as Multilayer Perceptron (MLP) or Support Vector Machine (SVM) were originally designed at the dawn of the ML era to produce a binary output in a form acceptable to computers. The design of MLP as Neural Network is purely based on the activation function (e.g., logistic) that either activates (state "1") or deactivates (state "0") the output neuron. There is almost no way to use such single-output NN architecture for multinomial classification. So, one must utilize either one network with multiple outputs or multiple single-output networks for the defined purpose, according to Aly [56]. Multiple outputs are normally used for the explicit determination of the class label or per class probability. Ou et al. [303] performed an extensive study of the multinomial classification by NN and concluded that the optimal class boundaries can be found when the method separates all classes at the same time by a single NN model. Also, the main problem with multi-output NN is in training with discarded relevant information from *"others"* classes. Such models are hard to train with large numbers of classes, while single-output models are easier to handle. On the other hand, SVM was originally designed for binary classification format. Therefore, many different schemes for multinomial classification were presented to supplement an intrinsic binary architecture, as for example the decision tree-based SVM proposed by Madzarov et al. [259]. One can also mention *One-against-One* and *One-against-All* approaches used to apply binary classifiers to multinomial problems. Both schemes are natural extensions of binary classifiers according to Avia-Herrera et al. [67]. The authors proposed the *One-against-All* logic analysis algorithm to handle multinomial classification since this scheme faces many problems due to imbalanced training. On small sets, it produces good results using a mixed-integer programming approach.

[38]The main ideas of this subsection are published under the contributions [372, 377]

Furthermore, *One-against-One* gives a better generalization since it eliminates the problem of imbalanced training, and allows only $\frac{N_C \cdot (N_C - 1)}{2}$ binary classification models for N_C classes rather than N_C models for the *One-against-All* scheme, according to Alvear-Sandoval et al. [55].

NF operates with a slightly different situation, where an activation function can be omitted and replaced by a defuzzification function, which is a real-valued one and is on the edge of classification/regression. The Mamdani-type NF model uses defuzzification to turn the fuzzy output value into a crisp value of some defined parameter that measures the fuzzy output. The most common application of NF is for the detection of something that is known to be *"good"* and something known to be *"bad"*. We went through research with NF classification that used IF-THEN rules, and found that most of them use a *"one-hot"* output encoding scheme. Chavdan et al. [363] used Mamdani-type rules for the multinomial classification problem of network attacks detection. Sindal et al. [391] presented an NF system for multinomial services in CDMA cellular network. There was an adoptive controller with a *"one-hot"* encoding scheme in the output layer. 4 outputs were used for each of the action classes. Furthermore, Yu-Hsiu et al. [253] proposed a novel NF classifier where every class was represented by a separate output, then encoded together with input parameters for particle swarm optimisation. Several multinomial classification problems were explored by Eiamkanitchat et al. [135] with respect to a novel NF-based method, where *"one-hot"* encoding was used as well. The same output encoding approach was applied by the Guo et al. [172].

As can be inferred, multinomial classification problems (especially in Digital Forensics) require additional optimization since they are more complex to deal with. Many of the existing models are only designed to handle binary classification problems.

Community-Accepted Classification Methods

Multinomial classification can be considered a harder task than binary in separating similar properties of similar classes due to the mix of parameters and possible values. From the book by Kononenko et al. [232] we can see that the majority of the supervised learning problems in Machine Learning are covered by binary classification methods. There are the following examples of originally-binary classifiers:

- *Decision Tree* uses a binary split of attribute values in the form of comparisons $>$ or $<$ on the way to next attribute in a tree. Decision nodes represent classes also as a binary split.
- *Support Vector Machine* builds a hyperplane that separates both classes using quadratic programming methods; however, kernel trick has to be applied

in case the classes cannot be separated linearly.

- *Logistic Regression* makes a classification in the form of binary-dependent variables that use logistic regression analysis to separate classes along either 0.0 or 1.0 on a logistic function.
- *Multilayer Perceptron* is a Neural Network-based method that maps n-dimensional input space into two asymptotically different values of the activation function, which could be a logistical one or something similar. It also has multiple hidden layers that model non-linear dependencies.

To extend the application of binary classifiers for multinomial classification problems, two ways of doing so can be named. First, there exist a number of modifications like *Multinomial logistic regression* designed for multiple categorical dependent variables. Second, the original method is preserved, while ensemble learning strategies are used. The main conventional strategies developed for such purposes are denoted by Allwein et al. [52] and Ou et al. [303]

- *One-against-One* trains a binary classifier for each of the N_C classes against every other class.
- *One-against-All* splits all classes in a way that every class from N_C is trained against all other classes except that one.
- *P-to-Q* selects a set of P classes that are going to be trained against Q other classes.

When it comes to purely multinomial classification methods however, there are several methods that are generally considered as designed for such problems according to Kononenko et al. [232]:

- *k-Nearest Neighbours* is a distance-based method that uses majority voting to classify new data samples based on previously known labelled samples.
- *Random Forest* is an ensemble learning method based on decision trees, which are generated using randomly selected attributes.
- *Naive Bayes* is a probabilistic classifier that calculates the chance that a particular set of attribute values appear in a particular class.
- *Bayesian Net* is a probabilistic classifier based on the conditional transition between attributes values.

General Problem Reduction & Output Encoding Strategies in Neural Network-Based Architectures

Chen et al. [96] highlighted that NN-based methods are generally capable of separating k-different classes at the same time. In reality, this means that multiple outputs have to be used. This is because the NN activation function is usually a differentiable continuous one, and can converge to two asymptotically different values such as in the case of a logistic function. Otherwise, reducing strategies have to be applied to utilize multiple binary classifiers with a single defuzzifier output, like *One-against-One* and *One-against-All*. As a result, there must be multiple NN models created for the multinomial classification problem, where each output is a binary $[0; 1]$. Considering this, the different output representation/encoding schemes were to be applied in Neural Network-based architectures as proposed by works by Aly [56] and Hurwitz [192] also shown in the Figure 2.27 and Table 2.2.

- **One-hot** such that *0001* (for 4 classes example) uses a single output flipped as *'1'* to denote a specific class, while other outputs are *'0'*, so the approach is stable against errors. The number of required outputs are therefore: $q = N_c$. This is the simplest naive approach used in works related to multinomial-classification problems. The drawback however is that it can be affected by imbalanced learning caused by assigning a single target class as *'1'*, while *other* classes are labelled as *'0'*.

- **Distributed-Output** or **Binary** methods such as *0101* uses a unique simple binary code (also could be different, like Hamming or error-correcting) to represent each class. The most commonly used name is also Error Correcting Output Coding (ECOC). Dietterich et al. [121] presented a ECOC study on different methods, where the efficiency of the method was proven to learn Decision Trees and Neural Networks on datasets partially represented in this work. Also in the problems where it is required to produce human-understandable models, ECOC cannot be applied due to its complexity. The total number of required outputs is lower than the number of classes in a problem and equal to $q = \lceil \log_2 N_c \rceil$. However, according to Hurwitz [192], this method is rather more susceptible to errors than *"one-hot"*, and can produce mistakes while a model is trained. Furthermore, to improve this method, Liu et al. [255] suggested the training of binary classifiers jointly rather than for separate sub-problems. This is done via a unified objective function.

Table 2.2: Example of Neuro-Fuzzy output encoding schemes for 4 classes

Class Label	Scheme	
	Binary	One-hot
Class '1'	00	0001
Class '2'	01	0010
Class '3'	10	0100
Class '4'	11	1000
Number of outputs	2	4

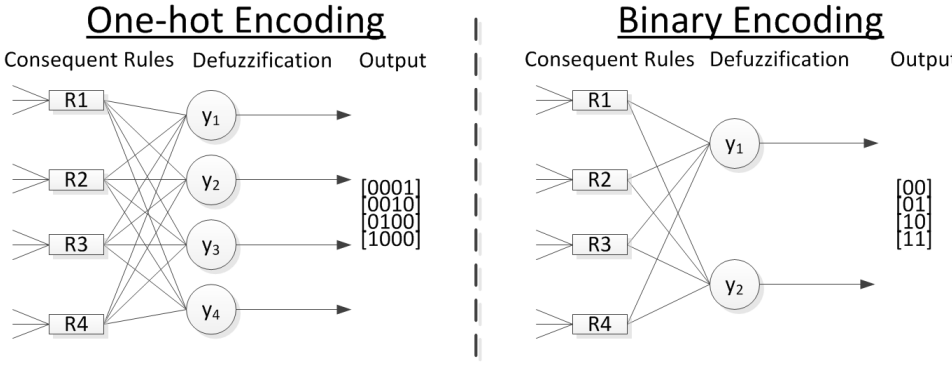


Figure 2.27: Comparison of Neuro-Fuzzy architecture with different output encoding schemes

Defuzzification in Multinomial Methods

Defuzzification is the final part of the NF method that converts a fuzzy value back into crisp value when necessary. *Takagi-Sugeno* regression model does not need defuzzification to extract the value, which is usually a function over the input features X. Despite this fact, a *Mamdani-type* classification model requires defuzzification, which is usually a time-consuming and complex process [95], in case a crisp value is needed rather than a fuzzy class label. It is an important task to convert a fuzzy value back to crisp numerical one. Many examples of defuzzification methods can be cited as studied by Naaz et al. [292]: Center of Gravity, Bisector of Area, Mean of Maximum, Smallest of Maximum. The authors performed an experimental study that showed that Center of Gravity can be considered as a method that produces better results in comparison to other methods. In the case of binary classifiers, it is easier to derive the continuous value of the output, which can be easily used with multiple-output coding. Yet it becomes more challenging when the system uses a single output. Kosko [233] suggested the use of an unequal-

weight center-of-gravity defuzzifier since.

$$y = \frac{\sum_{i=1}^{N_R} \cdot \mu_i(X) \cdot w_i \cdot V_i \cdot c_i}{\sum_{i=1}^{N_R} \mu_i(X) \cdot w_i \cdot V_i} \tag{2.39}$$

Since the volume V_i does not change and is influence by α however, it can be eliminated from the previous equation. Therefore, we are going to use equal-volume rules to eliminate the dominance of some rules over others.

$$y = \frac{\sum_{i=1}^{N_R} \cdot \mu_i(X) \cdot w_i \cdot c_i}{\sum_{i=1}^{N_R} \mu_i(X) \cdot w_i} \tag{2.40}$$

where X represents an unlabelled sample to be classified, N_R is a number of rules, extracted on the 1^{st} step and c_i denotes a centroid of a particular patch. Similarly, *Takagi-Sugeno* rules represent a function over the vector of antecedent values.

2.5.8 Higher Level of Abstraction & Deep Neural Networks[39]

There is a strong need for advanced computational models capable of high level abstraction modelling due to multinomial classification problems that usually involve a high level of non-linearity and complexity. We believe that the emerging area of Deep learning (DNN in particular) may facilitate this goal, as was mentioned before by Stallkamp et al. [400]. Previously, Ou et al. [303] performed an extensive study of the multinomial classification by general Artificial Neural Networks (ANN) and concluded that a single ANN model can achieve reliable classification accuracy. DNN however has been developed to tackle hardly-differentiable abstractions in data. Recently, Schmidhuber [357] performed a comprehensive overview of Deep learning in ANN. The majority of the state of the art Deep learning methods have been developed after 1990. The drawback of such a method is an inability to extract an exact meaning of the model, or derive a human-perceivable representation of a model through linguistic fuzzy logic rules. On the other hand, Fuzzy Logic has been used as a human-understandable model before, studied in depth by Zadeh in his works starting from 1960 [453]. Deep learning in NF was not mentioned in the work by Schmidhuber, so we believe that it might be a stepping stone for future research targeting understandable models in multinomial malware classification.

Complex data modelling in Neural-Architectures

ANN is a powerful method that is designed to handle non-linear dependencies in the data. It can be trained from data that are complex and non-linearly separable.

[39] Ideas of this subsection are publshed under the contribution [380]

An example is XOR problem where ANN is successfully utilized to predict an output from two inputs. Cottrell et al. [106] described the evolution of ANN's ability to handle complex data and studied the great potential of Multilayer Perceptions as well as SOMs, which are also used on the 1^{st} stage of NF. In some cases however better generalization is required to describe non-linear relations in the data. Alternatively, this can be done either by increasing a number of hidden layers that results in Deep Neural Networks (DNN), yet this is out of our scope due to producing a hardly interpretable model. Dahl et al. [108] presented a work on the application of three-layers ANN to classify around 200k samples into benign and malicious as well as into the 134 malware families using 4,000 features extracted from files. However, there is another way of keeping model understandable. Hybrid Intelligence methods such as NF are used to enhance the capabilities of ANN and to provide classification fuzzy rules.

NF is a two-stage method that includes the grouping of unsupervised samples and tuning of fuzzy rules, resulting in a rule-based fuzzy classification model [233]. The 1^{st} stage of NF, which influences the accuracy and robustness of the whole model, is based on Self-Organizing Map (SOM) learning. There has been work on Growing Hierarchical SOM [51] to improve clustering, but it is hardly applicable for NF since the fuzzy relationships cannot be represented properly because the boundaries of the different layers are crisp. As an analogy, we can look to Convolution Neural Networks (CNN) [357] which are used for visual pattern recognition. Yet this is out of our scope since sub-sampling may not produce understandable regions which can be explained by fuzzy rules later. The 2^{nd} stage of NF is fuzzy rules tuning to produce a reliable classification model. Many researchers have been using Deep learning to enhance existing ML algorithms. Below, we present the fundamentals of DNN as well as the limitations of the current state of the art related to Deep learning in NF for multinomial classification.

Importance of Deep Neural Network

DNN has been inspired by the era of Deep learning [357], and differs from the conventional definition of ANN by using higher levels of abstraction, or more hidden layers, in order to be capable of handling nonlinearities in the data. This was done to overcome the limitations of binary classifiers on non-linear problems, such as XOR among others [77, 286]. Moreover, the use of more hidden layers allows DNN a better approximation than conventional Soft Computing methods like Support Vector Machines (SVM). Also, one can see the large number of variations in how the architecture can be constructed due to the number of independent variables. The main parameter that defines the depth of DNN is a number of hidden layers used in a trained model. The *"golden rule"* is that ANN with more than *3 layers* can be considered deep, as also described by Karpathy et

al. [215]. At the same time, the *"rule of thumb"* defined by practical consideration is $(number\_attributes + number\_classes)/2$ used in Weka [149]. Some CNN tend to use at least 10 hidden layers in consideration of the need for iterative sub-sampling on a big image identification task [215]. Despite these advantages, DNN cannot be used to generate human-like solutions or, at least, to make an understandable representation of the model. Therefore, we believe that the synergy of DNN and NF is beneficial.

Deep Learning & Neuro-Fuzzy

There have been developed a number of Fuzzy Systems, including NF [38, 425] based on a specific task and required fuzzy rules configuration. Such systems however can be considered as linear since the abstraction in the data is incorporated only through the linear mixture of the membership functions. Non-linearity is an important requirement for being able to model multinomial problems according to Ou [303]. Originally, Neuro-Fuzzy [233] had been proposed as a synergy of ANN and pure FL to be able to extract fuzzy rules automatically without the involvement of a human expert. ANN offers high accuracy though the weights cannot be interpreted, while FL provides clear understandability, yet requires manual tuning of the parameters. Therefore, we have identified four research works that target deep learning specifically in relation to a Neuro-Fuzzy approach. In 1996, Tano et al. [411] presented Fuzzy Inference and Neural Network in Fuzzy Inference Software (FINEST). Even though the authors state that the system presents a deep combination of ANN and FL, it is in fact a hybrid system that consists of four layers: antecedent fuzzy sets, fuzzy rules AND combinations, output consequents, and a final layer that combines all rules outputs. Furthermore, a fuzzy deep belief network algorithm was proposed by Zhou et al. [462] in 2014 for sentiment classification. Its achieved accuracy was 66.6% – 75.3% on five different datasets with 2,000 samples each in binary classification problems (positive and negative reviews). It consists of an integration of Fuzzy Logic and DNN without the intermediate grouping step that was originally proposed by Kosko [233]. Instead, a Membership Function (MF) is presented for each of the input features in each class. The authors used 3-layer DNN with one output layer for features extraction, which is the minimal number of layers considered to be deep. Moreover, the authors did not consider the extraction of fuzzy rules, making this model suitable only for classification, and not for decision understanding. Following that, a Deep Cascade Neuro-Fuzzy System was proposed by Hu et al. [189] in 2016 to facilitate high-dimensional online fuzzy clustering. From the work however, it is not clear how the deep learning is used, though the fuzzy clustering procedure is explained in detail. Also, there was no mention of how the rules can be extracted for further use in practical applications. Thus, we cannot justify that this method can fully

be qualified as Deep Neuro-Fuzzy. Finally, Aviles et al. [68] states in 2016 the importance of combining deep learning and fuzzy theory to make a better Hybrid Intelligence method. The authors applied a combination of Deep Networks and Fuzzy Logic for handling visual uncertainty in robotic surgery. In particular, they modified Long-Short Term Memory networks that resulted in recursive deep learning. The regression-based estimation of the error showed improvement beginning at 35% to the accuracy rate. One may conclude therefore that the application of Deep learning for NF to enhance model accuracy and extract corresponding IF-THEN classification rules has not been sufficiently explored before. As a matter of fact, no work has yet explored the synergy of DNN with NF using more than 3 layers of architecture.

2.5.9 Challenges with Pro-Active Training of Neural Network-based Architectures[40]

The Big Data paradigm brings new challenges to data analytics. Below, we will consider several such obstacles that make utilization of conventional Machine Learning methods less efficient.

Dynamic Modification of Fuzzy Sets in Neuro-Fuzzy

Despite the flexibility of the Neural Network and the Fuzzy Logic, the Hybrid Intelligence system of NF is not modifiable and re-trainable. In the paper by Pitz et al. [313] it was stated that such systems lacks the ability to tune the topology. The KBANN and TopGen algorithms were studied and it was found that there is a need for intelligence-based growth in the topology rather than just automated updates. In [69], the dynamic node creation for ANN was proposed to show the possible retraining of the network without a loss in accuracy. The change in the topology of the Hybrid NF is more complicated since it binds the fuzzy rules selection, including the fuzzification part, inference mechanism, and defuzzification engine according to Fuller [155]. In this work, we concentrate instead on the change between the fuzzification part and the inference mechanism. Generally speaking, Neuro-Fuzzy is a Hybrid Intelligence method that assembles FL and ANN into a classification model [233]. The input data sample is a real-valued vector $X = \{x_i \in R, \leq i \leq M-1\}$, meaning a corresponding set of features $X = \{x_0, \ldots, x_{M-1}\}$, and it characterizes a point in M-dimensional space (number of features in input data). The mapping of input X into output Y is performed via the fuzzy patches [46]. Extension of the fuzzy set by adding a new term in to it will affect mapping since a fuzzy set L of the input X corresponds to a crisp set of classes C labels of the output Y. Thus the set L has the following form for all

[40]The main ideas of this subsection are published under the contributions [370, 368]

m terms in the fuzzy set using *one-input-one-output* topology:

$$L_i = \mu_L(x_1)/x_1 + \cdots + \mu_L(x_m)/x_m \tag{2.41}$$

where $+$ represents the notion of the logic operation OR in the domain of L. By means of the Zadeh's extension principle, the mapping can be expressed in a corresponding form [155]. According to the principle, this can be expressed as mapping from the input fuzzy set L into a class C using the transition function f:

$$C = f(L) = \mu_L(x_1)/y_1 + \mu_L(x_m)/y_m \tag{2.42}$$

In the malware detection problem, the input has to be constructed of multiple Universes X resulting in *multiple-input-one-output* a mapping scheme that gives spatial fuzzy patches on the multidimensional input space [322]. However, we have to apply the max principle since in this mapping scheme, there can be $x_1 = x_2$:

$$\begin{aligned} \exists\, x_i \in R :\ y = f(x_1) = \cdots = f(x_p) : \rightarrow \\ \mu_B(y) = \max_{y=f(x)} \mu_L(x_i) \end{aligned} \tag{2.43}$$

Therefore, the fuzzy inference principle is performed via the *max-min* principle for the Mandani-type rules based on the corresponding t-norm implication $x \rightarrow y = T(x, y) = min(x, y)$ [194]:

$$\begin{aligned} \mu_c(y) = \\ \max_{y=f(x_1,\cdots,x_m)} (min(\mu^1_{L_1} x_1, \cdots), \cdots, min(\mu^k_{L_m} x_1, \cdots)) \end{aligned} \tag{2.44}$$

where a number of fuzzy sets L_i is defined over the corresponding input set X. Generally speaking, this mapping establishes a relationship between the antecedent and consequent by means of fuzzy patches $\prod_{xy}$ with corresponding relationships between input and output. However, the introduction of new terms will affect the patches and split some of them into sub-patches. This will change the mapping function since new terms will be added in each of the linguistic variables L_i.

Fuller [155] stated that the cooperative models used for independent adjustment of each component of the Neuro-Fuzzy systems while hybrid models perform construction of the IF-THEN rules. Self-Organizing Maps and fuzzy clustering can be applied for rule learning in such cases [128, 355]. However, the cooperative system the requires a complete re-training of the neural network before the fuzzy component is initialized. When the rules are constructed and weights are calculated, the

rules voting process has to be completed according to Ishibushi et al. [200]. Also, it is important to select the smaller subset of rules that possess sufficient accuracy to simplify the model. This is due to the exponential dependency of the number of constructed rules on the dimensionality and number of the terms in each linguistic variable L_i. In the research [293], authors proposed to apply Genetic Algorithm for the selection of the most important rules. This method is stochastic and may produce different rules subsets for the same collected data. However, the most common method is to evaluate all the constructed rules. Yet there has been no proposed method of rules selection based on the weights importance and membership function.

Expansion of Fuzzy Sets In Trained Model

However, despite the applicability of Neuro-Fuzzy there exist challenges related to fast the emergence of new technologies. The problem of adding a new term into a fuzzy set of an already trained Hybrid NF model on malware detection problem can be formulated as follows: the NF model is learned from the data and is in the state S_0. After some period of time, there is a need to include a new $k-$th term L_i^k into an arbitrary linguistic variable L_i. Due to the possible limitations in the real-world with respect to repeated data gathering and retraining, the model has to be incrementally trained from the state S_0 with the change in topology to a new model in the state S_1. From the perspective of malware analysis, the models in both states have to be compatible with the collected characteristics of malware. For example, when a new feature is added to Android API, there is a need to expand the detection model and therefore preserve accuracy for the platform with and without this feature. This is one of the most widely used mobile platform OS, which has been in active development since 2008. As of January 2017, platform versions 1.0 through 25 are available (Android 7.1) [165], each containing new features and their included functionality. Since there has to be an initial reference model in the state S_0, we need to introduce the *transition* of the model $S_0 \rightarrow S_1$. By transition, we mean the modification of the model by adding newly extracted rules into the reference model. In this work, we use rectangular fuzzy patches $\prod$ that produces a change in NN weights when the terms are added in.

Generally, the number of fuzzy rules constructed during NF model training is K^M. Furthermore, adding a single new term in the input fuzzy set part of the NF topology will bring K^{M-1} newly constructed rules. Most important for the decision, the rules have to be extracted from both sets, so there will be no loss in accuracy. Additionally, the distribution of the input features does not significantly influence the model since the abstraction by the fuzzy logic is inserted. The main challenge however is to update the model in the state S_1 since it is on the edge of fuzzification, and the inference and change in terms will cause a change in the set of rules

and weights as result. There might be several causes of a need to expand the NF model without much retraining. Among them is a change of characteristic, for example on mobile phones with the Android platform when new types of data or new permissions are added. The sketch of the process of adding a new term is depicted in the Figure 2.28.

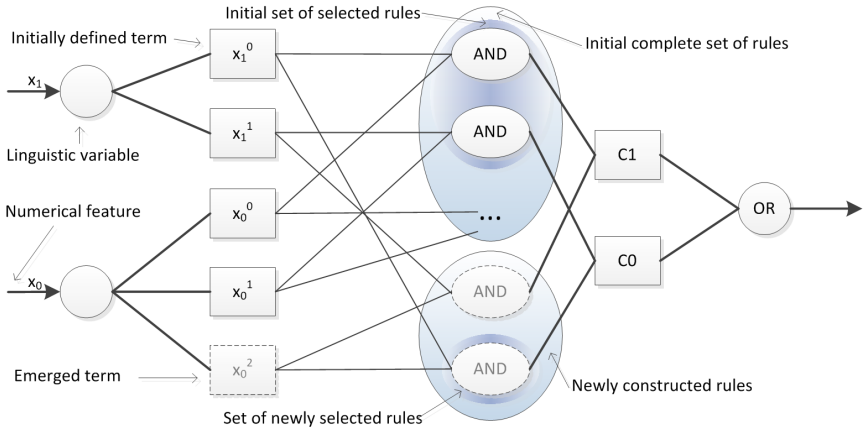


Figure 2.28: Dynamic expansion of a fuzzy set in Hybrid Neuro-Fuzzy with two classes: benign and malicious

There are two existing ways of dealing with changing the characteristics of features and fuzzy sets respectively. (i) The properties of the fuzzy terms can be adjusted, though the set will remain the same. This will mitigate the "concept drift" [320] (as in case with data streams), though not the need for fuzzy set extension. (ii) The NF model can be trained from scratch in order to apply the changes in the terms set, though it is not sufficient when the training delay is crucial. Thus, it must be clearly stated which model state is used to collect the data from the questioned application sample. Also, there is a need for an intermediate buffer of newly added rules.

On-line Training of Neural Networks

ANN is one of the most powerful ML methods capable of learning from erroneous, complex and incomplete data. As mentioned above, the training can be done either off-line or on-line. In this work, we target *on-line* learning since the model should be capable of adjusting the parameters of the model from data when a new sample comes.

Definition 1: MLP training is done via minimization of the objective function of

the error signals $E(W)$:

$$E(W) = \frac{1}{2}(y - d)^2 \tag{2.45}$$

where d - desired output of the MLP and y - actual output, and W is a set of all weights in MLP. The main obstacles in learning are that the method can get stuck in local optima unless the learning rate is an optimal one. So, the primary optimization problem in the MLP is minimisation of the function:

$$min\ E(W)\ |_{W \in \mathbb{R}^M} \tag{2.46}$$

where each function in the $E(W)$ is an objective function of the neuron's weights w_i^j ($j - th$ hidden layer and $i - th$ hidden unit) that should be optimized with the following condition on the whole domain of the function $dom\ E(W) = \mathbb{R}$, and with respect to the learning rates α:

$$\forall\ w_i^j\ \in\ \mathbb{R}:\ E(w_i^{j^*})\ <\ E(w_i^j) \tag{2.47}$$

According to Heskes et al. [184], *on-line* learning introduces a new challenge in adapting to new data that arrive. Saad et al. [344] presented an analysis describing the on-line learning within MLP, where the authors focused on the dynamic evolution of the error function. Recently, there has been research on the improvement of supervised learning in back-propagation MLP as well as the creation of new methods for it according to Heskes et al [184] and Mandic et al. [262]. Researchers have proposed different learning schemes, including complex algorithms like Widrow-Hoff LMS and Adaline as described by Widrow et al. [438]. The MLP learning process should be optimized as a differentiable error function, and then the Gradient Descent (GD) optimization of the function will tune the weight of the neurons:

$$w_i^{j^{new}} = w_i^{j^{current}} - \alpha \cdot \nabla E(w_i^j) \tag{2.48}$$

where $E(w_i^j)$ is a multidimensional error function over a weight w_i^j. The principle of the learning then is to use a so-called Delta Learning rule, comparing the output of the network against the labelled dataset. Delta Learning Rule makes a robust first-order approximation as stated by McClelland et al. [273] along the partial derivative direction only in case the learning rate is less than or equal to the optimal one, also described by Mandic et al. [262]. In real-world tasks however it is hard to predict how the optimal learning rate will change under the influence of an input concept drift.

The determination of the learning rate α brings with it the most challenge. There are several options for definition, such as constant rate or iterative adjustment [232].

However, the data sample are available for a short period of time and data flow has an unpredictable dynamic nature. Additionally, on-line training gives a faster convergence than batch training according to Wilson et al. [441]. Thus, the commonly used constant α is not suitable.

Error Back Propagation

The most commonly used method for ANN training is Error Back Propagation (EBP), which is based on the assumption that the error function $E(W)$ can be reduced by using gradient measure to find optimal weights.

Definition 2: The generalized EBP learning rule for back propagation is defined as follows, referring to GD method:

$$w_i^{j\,new} = w_i^{j\,current} - \alpha \cdot \delta_i^j \cdot g(h_i^j) \tag{2.49}$$

where α - fixed learning rate, $h_i^j = \sum w \cdot h$ - sum of the weighted signals from the previous neurons layer, $g(h_i^j)$ - sigmoid activation function of the neuron or value of the x_i in case the layer is the initial one. Moreover, depending on the layer, the value of the error signal will be calculated respectively for the intermediate layers:

$$\delta_i^j = \frac{1}{1+e^{-h_i^j}} \cdot (1 - \frac{1}{1+e^{-h_i^j}}) \cdot \sum w_i^j \cdot h_i^j \tag{2.50}$$

and corresponding output layer:

$$\delta_{output} = g'(h_i^j) \cdot (d-y) = \frac{1}{1+e^{-h_i^j}} \cdot (1 - \frac{1}{1+e^{-h_i^j}}) \cdot (d-y) \tag{2.51}$$

Originally, EBP makes it difficult to train each layer of MLP since the exact values of the output of each hidden layer are not known due to the fact that each layer learns more of the abstract meaning of the input data and the complexity of the model becomes higher with each new layer. So in general, the derivative of the error function $E(w_i^j)$ is calculated on each hidden layer.

The GD-based methods have been extensively studied recently. With respect to the optimization approach, there are several other valid ways of finding the optimal weights in MLP. The stochastic GD originally used for optimization has a low speed of convergence and is therefore the baseline. In order to improve learning, one may use the Conjugate Gradient Descent and Newton-Raphson as studied by McAllester [272], where Hessian matrices need to be evaluated. On the downside, this approach requires an additional algorithmic step to finding an optimal

direction and second-order derivatives, since the values of the derivatives are unknown and require multiple error functions evaluation during each epoch as result. On the other hand, a conventional MLP learning principle is not suitable for data streams mining since it provides an off-line learning over a given finite sample G. The stand-alone EBP method is highly resource-consuming in tasks related to Big Data in comparison with the speed of on-line processing.

Existing ways of α optimization for weights calculations

According to the literature review, there can be named several approaches to weight optimization in the Eq. 2.49: (i) static α with gradient information, (ii) iterative adjustments of α using optimization, (iii) higher dimensions factorization for faster convergence. In this work we consider the 2^{nd} option since it is less computationally expensive than 3^{rd} and more efficient than 1^{st}. According to Luenberger et al. [256], line searching along the gradient direction is the most promising method for iterative rate optimization since each weight can be optimized separately. In our view, this is the most promising approach in data streams mining, eliminating a need for expensive computations. Visibly, the learning rate α must either be defined empirically or use some one-dimensional optimization. According to Roy [342], in most of the related studies the authors used some predefined values or tried several empirical values. Since α is a constant in most cases is , it causes aggregation of the error and slow convergence when the value is not optimal. Generally speaking, the α can be optimized in different ways, such as the *adaptation of a learning rate* or *alteration of the gradient values*. Kandil et al. [214] proposed to use time-varying learning through the linearization of the whole network, requiring the additional expensive computation of matrices. This method uses a binary mapping scheme for computing the optimal learning rates for each of the layers. Yu et al. [451] described several approaches using first- and second-order derivatives for optimal α and momentum calculations, which are used to adjust the weights. At the same time, Tielman et al. [415] suggested the use of mini-batches and moving average of the squared gradient for each weight optimization. However, it requires keeping in memory a set of mini-batches with their corresponding moving average parameters in addition to a very slow alteration of a learning rate. Kingma et al. [227] optimized α using similar weighting on a momentum estimation. The authors suggested using decay parameters β, which require additional empirical estimations. Furthermore, Plagianakos et al. [314] suggested the use of pair-weights adjustments based on the previous epoch. On the other hand, one-dimensional optimization approaches like Golden Section Search (GSS) [256] can be considered a simpler and more effective approach than the application of Quadratic Programming. Therefore, GSS can be used in EBP to find an optimal rate α in an optimal region, rather than at some fixed starting point.

Definition 3: Golden Section Search is used to find the most promising value of α in the interval $[\alpha_{min}; \alpha_{max}]$ by means of following the iterative process with an error threshold ϵ:

$$\alpha_1 = b - \frac{b-a}{\phi},\ \alpha_2 = a + \frac{b-a}{\phi} \tag{2.52}$$

where initially $a = \alpha_{min}$, $b = \alpha_{max}$ and $\phi = \frac{1+\sqrt{5}}{2}$. The Equation 2.52 defines an iterative procedure, which is followed by the evaluation of $E(W - \alpha_1 \cdot \nabla E(W))$ and $E(W - \alpha_2 \cdot \nabla E(W))$. Each iteration ends with $a = \alpha_1$ or $b = \alpha_2$ until criterion $|a - b| > \epsilon$ is satisfied.

Another approach to optimization of α using EC was proposed by Kim et al. [224]. Some authors specifically target the generation of weights using EA rather than learning. Ding et al. [122] showed how the EC can be used for weights allocation. Jie et al. [208] highlighted the importance of GA in that it is able to produce a generalized model and improve learning. Furthermore, Islam et al. [419] specifically mentioned the applicability of GA to an optimized number of hidden neurons and layers towards the improvement of classification accuracy. Finally, Kanada [213] emphasized the importance of GA in learning rate optimization, where learning rate is increased or decreased with respect to previous epoch according to geometric regression. Another work of interest is by Sajan et al. [347], where authors optimized the number of nodes in addition to a single value of learning rate. In this work, we will target global optimization of each individual learning rate independent from previous values.

Chapter 3

The Proposed Soft Computing Algorithm for Digital Forensics Applications

This chapter is devoted to a novel Soft Computing Algorithm contributed by this thesis. In particular, we proposed improvements to the unsupervised 1^{st} stage of NF presented in Section 3.1, the supervised 2^{nd} stage, and towards multinomial NF and the corresponding defuzzification, which are given in Section 3.2. Our work towards Deep Neuro-Fuzzy is presented in the Section 3.3. Finally, we performed a comprehensive analysis of the proposed improvements in comparison to the Kosko method in the Section 3.5.

The Figure 3.1 shows the general steps in the Neuro-Fuzzy rule-extraction classification method [233]. It includes two stages: the unsupervised grouping of data pieces for fuzzy rules extraction and the supervised fine-tuning of fuzzy rules for building a precise classification model.

This method has not been used for Digital Forensics applications before. Moreover, during our work we identified a number of limitations and weaknesses in this method resulting in the (i) inability to handle large-scale data, (ii) incorporate such data properties as correlation and random distribution, (iii) incapability of performing multinomial classification.

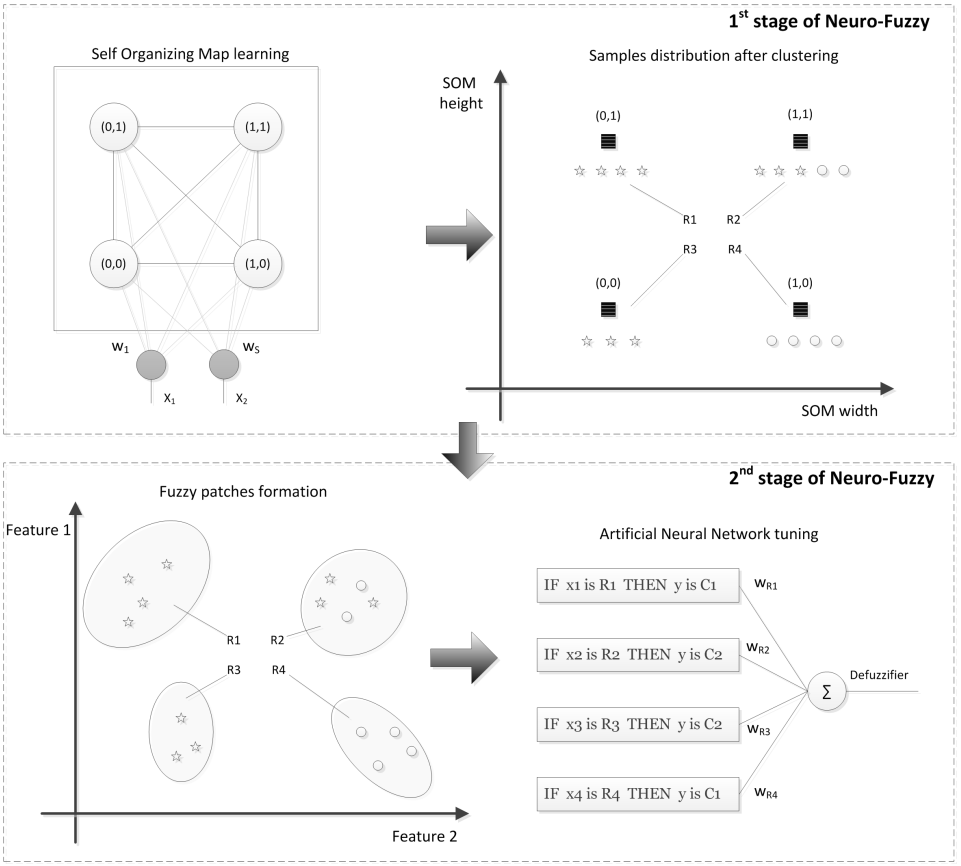


Figure 3.1: Neuro-Fuzzy approach that includes two stages [233]

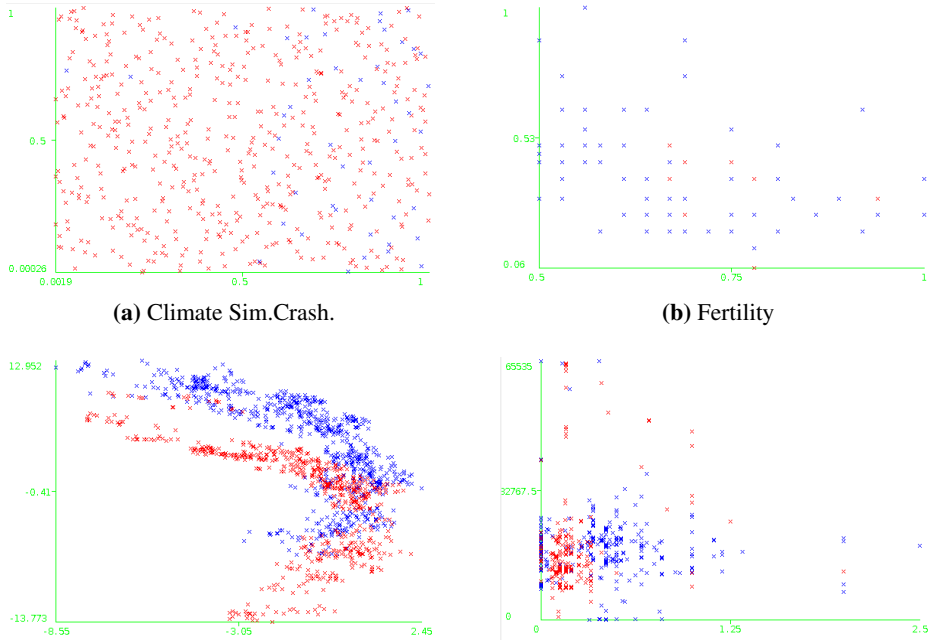
3.1 Neuro-Fuzzy Method - 1st Stage[1]

The proposed method revises the 1^{st} stage of Neuro-Fuzzy rule-extraction classification method, which is devoted to unsupervised learning via SOM grouping and fuzzy patch estimation from data as well as the impact of bootstrap learning on the overall complexity of the method. This stage results in a set of linguistic fuzzy rules that describe specific clusters found in data.

3.1.1 Inference of Self-Organizing Map Parameters

Here we provide insight into how the size of SOM can be determined using the data analytic. Exploratory data analysis provides a fast way of getting high level information about data properties. Considering this option, an automated proced-

[1]The main ideas of this section are published under the contribution [369, 373, 374, 375, 376]



(a) Climate Sim.Crash. **(b)** Fertility

(c) Banknote Auth. **(d)** Mob. malw.

Figure 3.2: Visualization of the dependencies between the features in 4 datasets mentioned earlier in Weka. The colors are blue and red denotes both classes

ure will be presented that gives an optimal size of SOM before it is initialized and learned, the main advantage being that no alteration will be made to the data. This is important since according to the Digital Forensics Process requirements [169], the data must be preserved without changing at all during the investigation process. As has been written before, the biggest eigenvalues ratio does not alone give sufficient information about the required number fuzzy rules.

Considering the previously mentioned obstacles and limitations for determination of the SOM size in Section 2, we use measures of non-linear correlation to find the best grid size with respect to the trade-off between interpretability and accuracy.

In the Figure 3.2 below, the visualization example of the dependency between the features in datasets are given. It can be seen that in case of independent features, the number of fuzzy patches can be decreased by covering more instances. From the other side, some non-monotonic dependencies require an increase in the number of fuzzy patches to reduce the error.

At this point, we can expect that the Banknote Authentication dataset might re-

quire the largest number of more specific rules to build the classification model, considering the non-linear and non-monotonic dependencies in Figure 3.2.c.

To deal with the uncertainty about the optimal SOM grid size, we apply the Pearson Correlation Coefficient (PCC) shown in Equation 3.1 as one of the factors enabling one to find the best grid size with respect to maintaining a reasonable balance between interpretability and accuracy, according to Shalaginov et al. [374].

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{3.1}$$

where x and y are two arbitrary variables with mean values $\bar{x}$ and $\bar{y}$ over a corresponding sample set.

Proposal 1: The measure of association between the variables can be used to determine the number of regions that will be clustered in SOM according to the Equation 2.12, using the absolute averaged value of PCC in the Equation 2.19. This means that the number of SOM nodes used to maximize $Accuracy\ (F)$ also depends on the data correlation.

$$S \propto |\bar{r}| \tag{3.2}$$

Remark. In real life applications, the data do not require the use of all rules with all combinations of the MF. Two contrary cases can be considered to show this. In the first one, data are random and attributes are uncorrelated, meaning that less specific fuzzy patches will fit the data better. In this case, the mentioned correlation metrics can be found in the Section 2, including $|\bar{r}| \to 0$. The second case is when the data are more deterministic and follow some patterns. This means that more fuzzy patches are required to describe the dependency specifically. Ideally, when the $|\bar{r}|$ tends to reach 1, more specific MF should be used to fit the data, rather than rectangular patches or even Kosko patches [233].

Lemma 1: The $Interpretability\ (F)$ in the Equation 2.12 can be formulated as a range of a number of rules that can be perceived by analysts without additional remembering efforts. Logical constraints are placed to limit the SOM size in order to make it understandable:

$$\begin{cases} S_{min} \geq N_C \wedge S_{min} \geq 2^2 \\ S_{max} \leq N_S^M \wedge S_{max} \leq 5^2 \end{cases} \tag{3.3}$$

where N_C is a number of classes in the classification problem, since at least such a number of rules are required to distinguish the data samples. $S_{min} = 2^2$ corresponds to the minimum SOM size used in GSOM [50]. $S_{max} = 5^2$ corresponds

to the maximal amount of the rules that can be used according to previous studies mentioned in the Chapter 2. A greater number of rules will create more difficulty in understanding. $S_{max} \leq N_S^M$ is the number of combinations of all MF in the rules.

Proposal 2: The optimal size of the SOM grid can be defined using the limitations mentioned before in the Equation 3.2, incorporating limitations from the Equation 3.3 via the degree of randomness α:

$$S_{Proposed} = S_{min} + \alpha \cdot (S_{max} - S_{min}) \tag{3.4}$$

Proposal 3: The degree of randomness in the Equation 3.4 is calculated using the mean absolute PCC as well as the metrics earlier denoted in the Equation 3.2:

$$\alpha = \frac{e_0}{e_1} \cdot |\bar{r}| \cdot N_C = \frac{e_0}{e_1} \cdot \frac{\sum_{i \neq j}^{M} |r_{ij}|}{M^2 - M} \cdot N_C \tag{3.5}$$

which is a subject to the following constraints:

$$0 \leq \alpha \leq 1.0 \tag{3.6}$$

where e_0 and e_1 are the 1^{st} and the 2^{nd} biggest eigenvalues that determine the components with the highest variance value. Next, the parameters of each of the 2D dimensions of the SOM are calculated using the Equations 3.4, 3.5, and 2.16:

$$S_H = \left\lceil \sqrt{S_{Proposed}} \right\rceil = \left\lceil \sqrt{S_{min} + \alpha \cdot (S_{max} - S_{min})} \right\rceil \tag{3.7}$$

$$S_W = \left\lfloor \frac{S_{Proposed}}{S_H} \right\rfloor \tag{3.8}$$

Remark. The modification of the *Vesanto* method was proposed in order to meet the interpretability requirements of the CF method. The proposed scheme does not depend on any number of instances since it is targeted at dealing with large-scale problems, as shown below in the Equation 3.9, in contrast to the Vesanto method specified in the Equation 2.13.

$$\lim_{N_S \to \infty} S_{Proposed} = S_{max} \implies \lim_{N_S \to \infty} S_{Proposed} \lll \infty \tag{3.9}$$

Moreover, the properties of the features such as normalization, number of features, constant values, and highly-correlated pairs will not influence the proposed method; this will be shown further on. The $|\bar{r}|$ is required as a mean measure for

a dataset, since only the average value of PCC is important, and not the direction (negative, positive).

Proposed algorithm for optimal SOM size in Neuro-Fuzzy training

Based on the studied literature and the problems with the predefined set of fuzzy terms, we propose the following optimization of the NF method based on the new estimation of Self-Organizing Feature Map size. It ensembles unsupervised clustering of similar applications and the tuning of extracted fuzzy rules.

Automated fuzzy rules training procedure based on elliptic fuzzy patches

1. Calculate all values of the PCC r_{ij} pair-wise between the attributes. Estimate the absolute mean of the PCC $|\bar{r}| = \frac{\sum_{i \neq j}^{M} |r_{ij}|}{M^2 - M}$.

2. Perform eigendecomposition of Correlation Matrix based on the earlier calculated PCC. Calculate the eigenvalues ratio $\frac{e_0}{e_1}$.

3. Corresponding H and W dimensions of the SOM are derived from the measure $S_{Proposed}$ defined earlier.

4. Clustering based on the features similarities is done by trained SOM to convert M-dimensional feature vector into $2D$-lattice that consists of $H \times W$ nodes. After training each node, $S_{i,j}$ it includes clustering of similar data samples as depicted in the Figure 3.3. One can see that a single SOM node can contain samples belonging to one or more clusters that characterize a particular class (samples of A or B classes in the Figure).

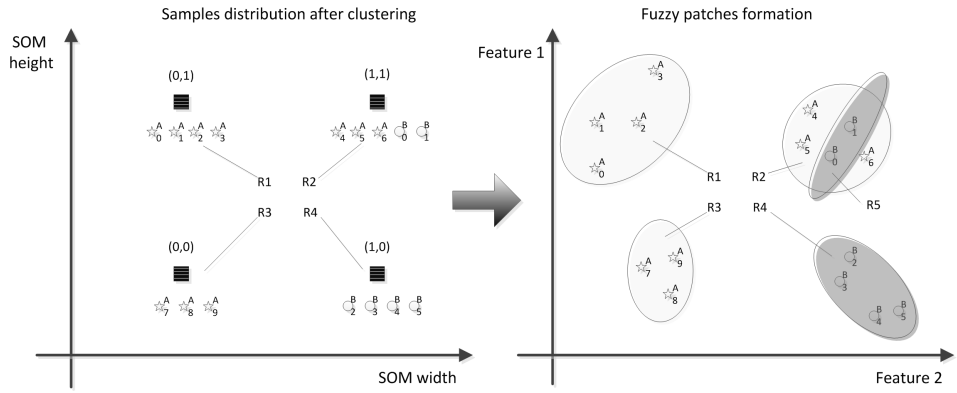


Figure 3.3: Extraction of elliptic fuzzy patches from trained Self-Organizing Map

The full description of the proposed procedure is given in the Algorithm 1. It is divided into four distinct functions. Line 1 describes the assignment of initial parameters necessary for the proper operation of the algorithm; where α_{χ^2} is a pseudo-radius used in elliptic fuzzy patches calculation, $SOMepochs$ describes the number of epochs in which SOM is trained at the 1^{st} stage of NF, $NFepochs$ - the number of iterations by which ANN is trained at the 2^{nd} NF stage, and BA is a fraction of the data samples from the initial dataset used in bootstrap aggregation.

The function on Line 2, $OptimalSize()$, estimates the optimal size of SOM according to properties of given data $\bar{x}$, and Proposal 2 & 3, deriving the optimal weight and height of the SOM grid as a result. The function invoked on Line 3, $SOMtraining()$, describes the conventional iterative training of SOM using BMU with respect to previously defined parameters. The outcome of such training is a set of clusters for each of the classes, i.e. a set of data samples at each BMU. These result in fuzzy patches to be used on the 2^{nd} NF stage.

Algorithm 1 Proposed way of training on the 1^{st} stage of Neuro-Fuzzy method

1: $SOMepochs \leftarrow 100$; $\bar{x} \leftarrow data$; $BA \leftarrow 1\%$
2: $Hgrid, Wgrid \leftarrow OptimalSize(\bar{x})$
3: $clustersConfig \leftarrow SOMtraining(\bar{x}, Hgrid, Wgrid, BA, SOMepochs)$
4: **function** OPTIMALSIZE($\bar{x}$)
5: $S_{min} = 2^2$; $S_{max} = 5^2$; $nC \leftarrow 2$
6: $Corr(\bar{x}) = PearsonCorrelationCoeffMatrix(\bar{x})$
7: $E0, E1 \leftarrow Eigendecomposition(Corr(\bar{x}))$; $alpha \leftarrow \frac{E0}{E1} \cdot avg(Corr(\bar{x}))$
8: $S_{opt} \leftarrow S_{min} + (S_{max} - S_{min}) \cdot \alpha$
9: $Hgrid = \left\lceil \sqrt{S_{opt}} \right\rceil$; $Wgrid = \left\lfloor \frac{S_{opt}}{Hgrid} \right\rfloor$
10: **return** $Hgrid, Wgrid$
11: **end function**

3.1.2 Fuzzy Patches Estimation

The method for elliptic patches configuration suggested by Kosko [233] does not provide a proper incorporation of the information from the elliptic regions.

Theorem 1: The problem of estimation of the elliptic fuzzy patch Π^i parameters for real-data clusters can be reformulated as a parametric distribution χ^2 test of fitting data into a particular distribution model [373].

At this point, we make the assumption that the data are normally distributed, and that the cluster derived from SOM will fit the elliptic region according to

Kosko [233]. The χ^2 test is designed to measure how well the distributed data fits Gaussian distribution, which has to be preliminarily validated when dealing with real world data. Since this test is originally designed for categorical data, we have to use χ^2 test for the variance as described in the book [263][Chapter 12], for $N_F - 1$ *df* transforms into an equation for the M independent random variables:

$$\chi^2 = \frac{(N_F - 1)S^2}{\sigma^2} = \sum_{i=0}^{N_F-1} \left(\frac{x_i - \bar{x_i}}{\sigma_i}\right)^2 \tag{3.10}$$

where $\bar{x_i}$ is a center of a particular cluster and σ_i is a spread around the center. *Proof:* By considering the non-transformed unfolded equation of the hyperellipsoid, we will get the following equation:

$$\frac{(x_0 - c_0)^2}{\sigma_0^2} + \ldots + \frac{(x_{N_F-1} - c_{N_F-1})^2}{\sigma_{N_F-1}^2} = \alpha^2 \tag{3.11}$$

which is the same as the following, considering one feature as a fixed variable:

$$\sum_{i=0}^{N_F-1} \frac{(x_i - c_i)^2}{\sigma_i^2} = \alpha^2 \tag{3.12}$$

The χ^2 test makes it clear that the parameter α^2 can be estimated from a contingency table, and is equal to the value of χ^2 for a particular confidence value β and a defined number of df:

$$\sum_{i=0}^{N_F-1} \left(\frac{x_i - c_i}{\sigma_i}\right)^2 = \alpha^2 \approx \chi^2|_\beta \tag{3.13}$$

The sample variance S^2 can be treated as variance of all elements in the particular data cluster, and standard deviation σ^2 as a theoretical deviation in this cluster, so we can state that $\chi^2 \approx \alpha^2$ with some degree of the confidence interval β. This challenge is related to the chance of outliers rather than fuzziness, as was explained by Ross et al. [339]. By introducing β, we are able to control the chance of outliers in data distribution, so as to avoid the possibility entirely.

The Figure 3.4 sketches the differences between the simple rectangular elliptic constructed by Kosko and proposed way of elliptic patches construction.

This definition of the elliptic region complies with the fitness of data within an elliptic region that may eliminate outliers or error values. In case of a significant

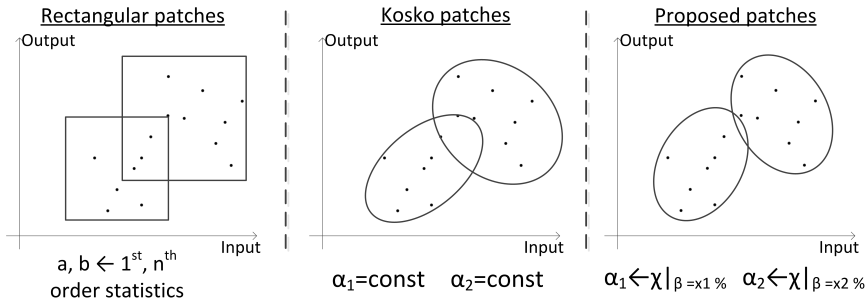


Figure 3.4: Examples of patches configuration: A simple, Kosko and proposed method

number of rules, this should reduce that number to get more specific fuzzy regions, and reduce uncertainty by overlapping regions.

To summarize, we use a χ^2 test to find the value of the parameter α^2 from the Kosko method. As was proven before, we come to the conclusion that $\chi^2 = \alpha^2$ in a defined confidence interval β with specified df for the statistical model of the data. This gives an adaptable model that adjusts the configuration of elliptic patches according to specified qualities of data that influence the selection of β.

3.1.3 Bootstrap Learning for Generalization

In order to improve the generalization of the method, as well as improve accuracy by reducing the overfitting, we apply bootstrap aggregation. At this point, we can hypothesize that bagging can help to reduce the errors due to outliers in the clusters associated with each SOM node. Dudoit et al. [131] studied the influence of bagging on the clustering procedure. This work can be considered relevant since we are working on the improvement of SOM clustering and of the generalization of the extracted patches. Dudoit et al. [131] performed a study of the clustering in new tumor classes' detection. They concluded that the application of bagging produced results at least as accurate as those without bagging. Since we are dealing with large-scale datasets, this study suggests bagging might not only result in improved accuracy, but also in a shortened SOM learning time.

Lemma 2: Bootstrap aggregation is performed to extract h of the random samples D_l from the dataset, equal to 1%. The best result is used to train the 2^{nd} stage of the NF method.

Remark. On the 1^{st} step of NF, the SOM is trained from the best selected D_l. Then, fuzzy patch parameters are used to train the ANN on the 2^{nd} step of NF. Such a configuration will be more robust than the training of ANN on the 2^{nd} step only using samples from the D_l.

3.2 Neuro-Fuzzy Method - 2nd Stage[2]

Below, we present an improvement targeted at the 2^{nd} stage of Neuro-Fuzzy rules-extraction classification methods that is devoted to fuzzy rules tuning in order to achieve a better performance of the classification model.

3.2.1 Membership Function Construction

The triangular MF used in the simple rectangular and Kosko methods are not appropriate for this purpose since they do not incorporate all available information from a constructed elliptic fuzzy patch. Moreover, the function should count on the rotation of the patch and distance from the center.

Theorem 2: A radial basis Gaussian MF can be used instead of the triangular projection-based MF to provide a better fit for the data in rotated elliptic fuzzy patches when the rules minimum combination is calculated, according to Shalaginov et al. [373].

Proof: The minimum principle used to define a rule's MF according to Dickerson et al. [120] is a combination of the following form of Cartesian products:

$$\mu_R = \mu_0(X) \wedge \mu_1(X) \cdots \wedge \mu_{N_F-1}(X) \tag{3.14}$$

where each MF function μ_i of each feature is defined as a triangular one. At this point, we replace the triangular MF by means of the Gaussian function for the feature i:

$$\mu_{a_i} = s_i e^{-\frac{1}{2}\left(\frac{x_i - c_i}{\sigma_i}\right)^2} \tag{3.15}$$

where $s_i \in (0; 1]$ is a scaling constant and other parameters are the corresponding statistical properties of ellipsoid projections on each feature space. Furthermore, this is used in the rule's MF in Equation 3.14 and for the overall MF in Equation 3.16. According to Zhang et al. [328] and Qiu [327], the overall membership grade is described in consideration of the collection of different image bands. The authors propose the use of M-th root to derive the overall MF. However, this will create a significant overlap between the rules that cause overfitting of the classification model.

$$\begin{aligned} \mu_R &= s_0 e^{-\frac{1}{2}\left(\frac{x_0 - c_0}{\sigma_0}\right)^2} \cdots \wedge s_{N_F-1} e^{-\frac{1}{2}\left(\frac{x_{N_F-1} - c_{N_F-1}}{\sigma_{N_F-1}}\right)^2} = \\ &= \prod_{i=0}^{N_F-1} s_i e^{-\frac{1}{2}\left(\frac{x_i - c_i}{\sigma_i}\right)^2} = \left(\prod_{i=0}^{N_F-1} s_i\right) \cdot e^{-\frac{1}{2}(x-c)^T(x-c)} \end{aligned} \tag{3.16}$$

[2]The main ideas of this section are published under the contribution [368, 373, 376, 377, 378, 381]

The scaling factors s_i are not known and have to be defined empirically. However, we consider the product of Guassian MF functions in Equation 3.14 as the multivariate distribution. From the other side, Kim et al. [225] introduced a Gaussian sum approximation as a product of a singular Gaussian MF. As a result, we make the scaling factors s_i equal to 1.0 because the rule's MF should not be restricted to a magnitude of $\frac{1}{\sqrt{(2\pi)^{N_F}|\Sigma|}}$ in the multivariate probability density function according to study [383], as it is used by Kim et al. [225]. Piegat [312] mentioned that the angle between the axis of the hyperellipsoid and its features can possibly help to increase the precision of the MF. This book presented an example of 2-D Gaussian MF that also incorporates an angle α. The author mentioned that such a model will have 5 degrees of freedom $(x_1, x_2, c_1, c_2, \alpha)$ for a two features model in comparison to a non-rotatable function, which therefore may become an obstacle to using it. Yet the set of angles from Equation 2.34 is already known, which does not require any additional computation steps. So, the generalized equation of the hyperellipsoid is then used in the derived Gaussian MF sum approximation that incorporates all available information from the elliptic region by means of a covariance matrix:

$$\mu_R = e^{-\frac{1}{2}(x-c)^T \; P \, \Lambda \, P^T \; (x-c)} \tag{3.17}$$

We have presented how the triangular MF is replaced by a modified Gaussian function to form a rule MF. This will better fit with the data when the features are correlated, as triangular MF based on the hyperellipsoid projections cannot cover the model properly. The differences between the three methods are presented in Figure 3.5.

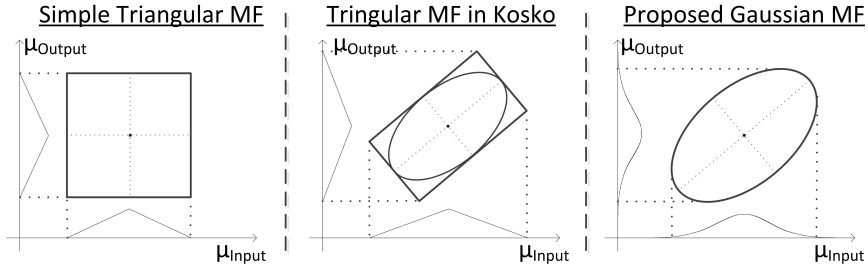


Figure 3.5: Examples of MF in simple rectangular, Kosko and proposed methods

3.2.2 Improved Multinomial Classification

Considering the fundamentals mentioned in the Section 2, the following challenges can be highlighted: in the case of *"binary"* and *"one-hot"* encoding, each weight set represents a different model that is usually trained with (1) major bias in the

number of other classes vs the targeted one. As a result, it also requires (2) training overhead when the number of classes is large. Following this, (3) additional weighting of the output in voting scheme might be necessary. Finally, (4) the used Center of Gravity defuzzifier also has to include class information. To overcome these obstacles, we suggest improvements that may facilitate the training of a multi-class NF.

Proposal 4: The grouping results of the NF 1^{st} step, i.e. SOM training has to be bound to produce statistically-sound parameters G for each fuzzy rule R_i:

$$R_i = G\{\ SOM\ \}|_{\substack{N_{h,w}^{Class_i} \geq \eta^{Class_i} \\ N_{h,w} \geq \eta}} \tag{3.18}$$

where $N_{h,w}$ denotes the number of samples allocated in a SOM node after clustering, $N_{h,w}^{Class_i}$ shows the number of samples in this node for each particular class $Class_i$, η denotes a minimal number of samples in each node to be eligible to extract the parameters of fuzzy rules, and η^{Class_i} denotes a necessary minimal amount of samples per class to be able to form a rule for a particular class $Class_i$. We believe however that while growing number of samples, the η has to be grown as well to eliminate that outliers influence classification. In contrast to the previous study [372], we believe that η^{Class_i} is too small to be used for a large-scale dataset. In spite of this, we gave the following empirical estimation for large-scale datasets.

Remark. After training SOM on the 1^{st} stage of NF, each node may contain many samples from various classes that result in a set of fuzzy rules described by those samples. Respectively, a fuzzy rule represents a set of statistical parameters of a group of similar samples grouped by every node in a SOM grid $SOM_{h,w}$ [233]. Therefore, there should be some minimal number of samples to derive a more generalized fuzzy model rather than a specific outlier one. *Rectangular patches* require at least two points to be able to circumscribe a rectangle and calculate the lengths. *Elliptic patches* may require at least three points for better describing the statistical properties. Thus, this will going to ensure the reliability of the derived rules from each node.

Proposal 5. To comply with the minimum number of data samples extracted from each SOM node, and to limit the growth of this number, the following estimation is given for multinomial classification problems:

$$\eta^{Class_i} = \sqrt{\frac{size(D_l)}{N_C}} \tag{3.19}$$

where N_C is a number of classes $\eta \geq 3$. This is done for better distribution of class samples grouped in a SOM node.

Remark. After training SOM on the 1^{st} stage of NF, each node may contain many samples from various classes that result in a set of fuzzy rules described by those samples. Elliptic fuzzy patches require at least 3 data samples for an elliptic equation. However, in the case of large-scale multinomial datasets, the minimal number of data samples must be higher to avoid errors due to outliers. This way, this step can ensure reliability of derived rules from each node.

Proposal 6: Use a single-output ($q = 1$) defuzzifier to avoid building a multi-output model as shown in the Figure 3.6. Thus, one can present training classes to the desired output pattern d_i during the 2^{nd} phase of the NF training process as numerical values replace the value of the centroid in the Equation 2.40:

$$c_i = d_i \tag{3.20}$$

where the class label d_i denotes a nominal natural value of the input sample, making it more consistent with Mamdani-type rules than with function-based Takagi-Sugeno.

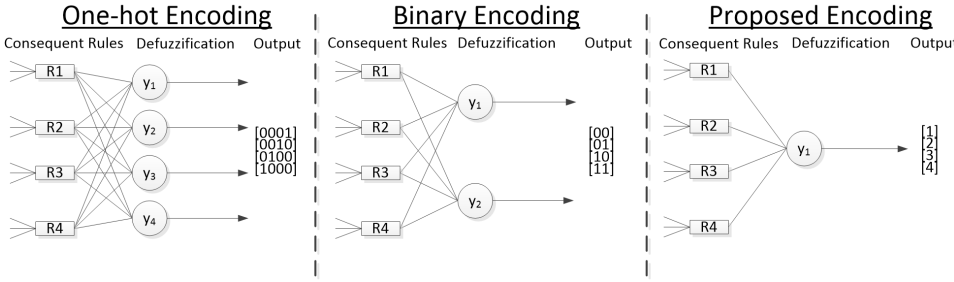


Figure 3.6: Comparison of output encoding schemes for Neuro-Fuzzy

Remark: The motivation behind this proposal is to reduce the number of output weights that have to be trained in contrast to multiple-output NN. It can be noticed that with a large number of classes, the process of training is affected by the Curse of Dimensionality. The Table 3.1 compares the output schemes.

Proposal 7: A special approach for defuzzification function calculation has to be used to comply with *Proposal 5* as described earlier [372]. The Center of Gravity fuzzifier used in work by Kosko [233] needs modification to incorporate more information as a single output in the NF model. We suggest the following one based

Table 3.1: Different NF output encoding schemes for 4 classes example

Class Label	Scheme		
	Binary	**One-hot**	**Natural**
Class '1'	00	0001	1
Class '2'	01	0010	2
Class '3'	10	0100	3
Class '4'	11	1000	4
Number of outputs	2	4	1

on the numerical class label:

$$y = \frac{\sum_{i=1}^{N_R} d_i \cdot \mu_i(X) \cdot w_i}{\sum_{i=1}^{N_R} \mu_i(X) \cdot w_i} \tag{3.21}$$

where N_R is a number of rules extracted from data on the 1^{st} stage of NF. In this case, we consider the natural number component d_i as the output, specified by the rule's consequent part, yet also compliant with the Mamdani-type rules. The scheme of the defuzzifier is represented in the Figure 3.7.

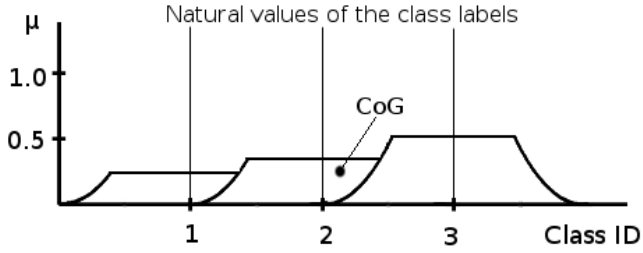


Figure 3.7: Center of Gravity defuzzifier using natural value of the Class ID label

3.2.3 An Insight into Dynamic Expansion of Linguistic Terms Set

The prospective challenge that may appear in proactive malware detection and computer system defense is the possibility of a change in the characteristics or properties that make an already trained model less efficient. This cannot be handled simply by means of adjusting statistical characteristics of the fuzzy set that mitigates the "drift of concept". The delays on re-training of the data model may have a dramatic effect of the automated incidence response. In spite of this, incremental learning of the changed NF topology has to be performed. This section provides a description of our method for the mitigation of the defined problem.

Inspired by the Rete algorithm [127], we propose a flexible model of the transitional fuzzy rules storage. First, γ-memory contains the initial set of the fuzzy rules. Second, δ-memory serves as a buffer that aggregates newly collected and selected fuzzy rules after a term in a fuzzy set is added. As a Hybrid Intelligence system, the *Type-1* Neuro-Fuzzy topology is used that has weights and outputs of the systems as crisp values and inputs as fuzzy attributes [155]. Hybrid NF gives a model that constructs rules, selects the most relevant, and then evaluates the set of the rules with respect to firing strengths that are defined as membership functions. Below, we consider the following sets of rules: the initially constructed set of fuzzy rules *RC*, the selected set after the initial training *RS*, the newly constructed set *NC*, and the set with selected rules from a new *NS*. The dataflow is given below:

1. The initial set RR of classification fuzzy rules at the NF state S_0 will be as follows, considering initially created rules:

$$\gamma = RR|_{S_0} \; = \; RC \tag{3.22}$$

2. After introducing new linguistic terms, a new set of fuzzy rules will be created:

$$\delta = RR|_{S_1} \; = \; NC \tag{3.23}$$

   According to Zadeh's extension principle, adding new terms will have an influence on *max-min* in fuzzy inference. At this point, a single-step incremental training would be preferable for adjusting the weights of new rules since the other part of the model is already trained. This means that the method keeps already learned rules' weights constant while adjusting the weights of the newly added rules. This helps to avoid complete retraining because properties of binary files do not change very often, and are rather affected by polymorphism. However, as the model is retrained, a considerable amount of properties are changed to keep it up-to-date.

   Considering this, we propose a rules selection method that is more appropriate for the specified conditions. Out of all the constructed rules, only relevant fuzzy rules with high accuracies should be considered. The rules with higher/lower weights in the NF are treated as more important since binary classification problems include both directions of importance: negative weight for one class and positive for another. This happens because the output of the network is as follows:

$$O_{RC} = w_1 \cdot R_1 + \cdots w_m \cdot R_m \tag{3.24}$$

where the change in fuzzy rules weights will cause a significant change in a rule's output for the corresponding class.

3. Finally, on the NF state S_1, considering logical disjunction of the fuzzy rules sets in both γ- and δ-memories, the set of fuzzy rules is as following:

$$\gamma \vee \delta = RR|_{S_1} \; = \; NS \vee RS \tag{3.25}$$

The conventional NF learning process consists of 5 stages, starting from data pre-processing and ending with Inference and Decision Making. The Figure 3.8 presents the new steps that are added to conventional NF.

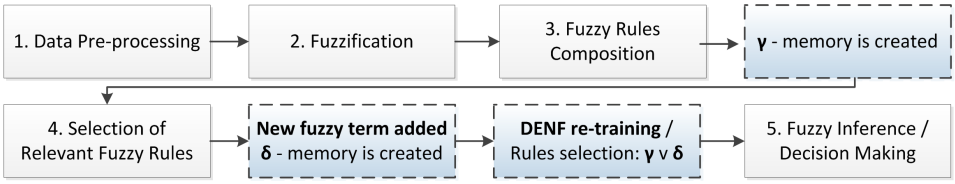


Figure 3.8: Conventional 5 stages of NF learning and proposed DENF stages (boxes with dotted lines)

In the [198] it was stated that a higher value of the weights brings a higher coverage of the classification for each class. Hence, we proposed the algorithm called Dynamically-Expanded Neuro-Fuzzy (DENF), which is denoted in the Listing 0 to encounter the challenge of the expanding fuzzy sets for the multi-valued logic.

In order to reduce the training error, we keep the initial weights values constant and adjust only the values of new weights in order to reduce the error function of the neural network $E(w)$. Additionally, this will decrease the time complexity that is beneficial for the application in critical infrastructure that requires learning from the large data sets. After the model is trained and the rules selection is performed, the search for a closest rule is done using the Euclidean distance between the sample's assigned linguistic terms and the selected rules: $\underset{RR|_{S_1}}{min} \sqrt{(\sum_{k=1}^{M} L_i^k - L_{assigned}^k)}$. Once the closest fuzzy rule is found, classification is performed using the min-max principle for both classes.

Furthermore, the comparison of the proposed method's computational complexity with the conventional NF without rules selection is presented in the Table 3.2. The NF are trained mE epochs over the total number N of labelled data samples. In this case, we consider the dependency of the number of arithmetic operations mainly dependent on the number of fuzzy sets M and number of fuzzy terms K. It can be seen that NF's initial training and retraining have the same complexity of

Algorithm 2 Dynamically-Expanded Neuro-Fuzzy (DENF) method for adding new terms in fuzzy set without complete retraining of the NF

$M \leftarrow numberFuzzySets$; $K \leftarrow numberTerms$
$mR \leftarrow numberSelectedRules$
$trainANN()$; ▷ Learning from the labelled data
for all for all fuzzy sets L **do**
    for all for all terms in fuzzy set L_i **do**
        $R[i][statement] \leftarrow L_1^1 \wedge L_2^1 \wedge \cdots \wedge L_M^K$
        $weightR[i][weight] \leftarrow w$
    end for
end for
$sortRulesImportance(R[all][weight])$; ▷ Sorting all the constructed rules
while n<mR/2 **do** ▷ Selection of important rules
    $R_{class1} \leftarrow R[n]$; $R_{class2} \leftarrow R[K^M - n]$; $n++$;
end while
$\gamma - memory \leftarrow R_{class1} \vee R_{class2}$ ▷ γ-memory composed from the rules
$newTerm \leftarrow idFuzzySet$ ▷ New term is added to a fuzzy set
$updateWeights()$; ▷ Retraining ANN for new rules (updating the weights)
for all for all fuzzy sets L **do** ▷ Retraining ANN except the expanded set
    for all for each term in fuzzy set L_i **do**
        $R^{new}[i][statement] \leftarrow L_{newTerm}^{K+1} \wedge L_1^1 \wedge L_2^1 \wedge \cdots \wedge L_M^K$
        $weightR^{new}[i][weight] \leftarrow w$
    end for
end for
$sortRulesImportance(R^{new}[all][weight])$;
while n<mR/2 **do** ▷ Selection of important rules from new
    $R_{class1}^{new} \leftarrow R^{new}[n]$; $R_{class2}^{new} \leftarrow R^{new}[K^M - n]$; $n++$;
end while
$\delta - memory \leftarrow R_{class1}^{new} \vee R_{class2}^{new}$ ▷ δ-memory composed from the selected rules
return γ, δ

$O(K^M)$ as well as the complexity of DENF initial training. This can be explained by the fact that the method is trained from scratch and the assumption is made that the data is new, and further that the re-training will take a lower number of arithmetic operations for DENF with complexity $O(K^{M-1})$ since the proposed improvements allow it to be re-trained using the previous NF model. The parallel time complexity is affected by the number of execution threads.

Method	Number of Arithmetic Operations	Sequential time
NF initial training	$N \cdot mE \cdot K^M$	$O(K^M)$
NF retraining	$N \cdot mE \cdot (K^M + K^{M-1}))$	$O(K^M)$
DENF initial training	$N \cdot mE \cdot K^M + K^{2 \cdot M} + mR$	$O(K^M)$
DENF retraining	$N \cdot mE \cdot K^{M-1} + K^{2 \cdot (M-1)} + mR$	$O(K^{M-1})$

Table 3.2: Complexity comparison of the proposed method and conventional re-training of the Hybrid NF for adding a single term in a fuzzy set.

It can be seen, however, that there is an exponential dependency on the number of terms in each fuzzy set on the time complexity. With the constant change in fuzzy terms, retraining will be time consuming. Also, for the purposes of fuzzy rules selection we apply the metric of importance of the rule that is roughly semantically close to the weight value. Moreover, the proposed method targets adding new rules, while a new property to the malware characteristic is added. However, deletion of the rules is out of scope of this thesis and will be considered in future research.

3.3 Deep Neuro-Fuzzy Architecture[3]

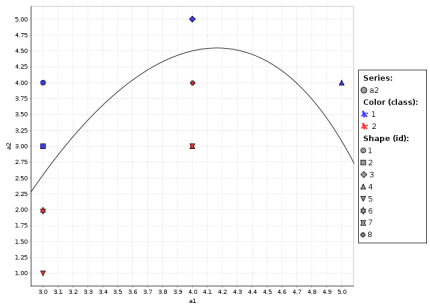
From the literature study, we found that NF has great potential in Digital Forensics, yet is hardly suitable for such tasks as multinomial malware categorization due to its shallow architecture. Another idea is the extraction of relevant fuzzy rules, and not the whole set of combinations of fuzzy set terms in contrast to [68, 117, 462].

The main drawback of the classical NF approach defined earlier by Kosko [233] is the shallow architecture that makes it impossible to model non-linear data. If we look at the available NF architectures [38], we notice that they have similar limitations, making such systems hardly applicable to multinomial classification of complex large-scale data. This shallow architecture means that the data regions defined by fuzzy sets are then weighted with the help of a linear output combiner of a form $y = \sum(w_i \cdot R_i)$ over a set of extracted fuzzy rules R. On the other hand,

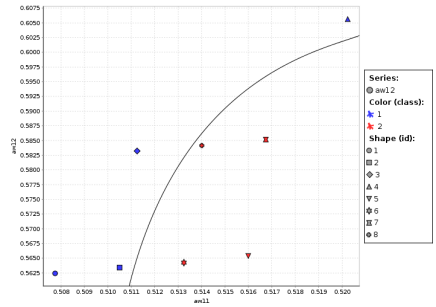
[3]The main ideas of this section are published under the contribution [380]

there are approaches like kernel methods in linear classifier SVM for example that increase the dimensionality of the problem being solved. Still, SVM is considered to be an intrinsically shallow algorithm, which makes it impossible to achieve high classification performance on many-class problems [377].

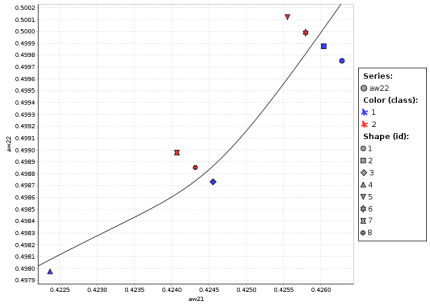
To the contrary, DNN is capable of feature space transformation in such a way as to make it separable [77]. With more hidden layers, one can achieve better class separation due to feature space transformation into other representations. Inspired by [301], we studied how DNN is better when it comes to non-linear data as presented in the Figure 3.9. This is a simple example of 3-layer DNN showing how the class boundary changes in each of the hidden layers together with PCC value r.



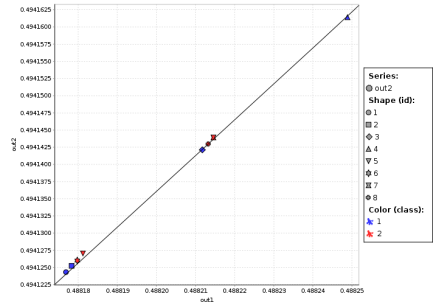
(a) Representation on the input layer: $r(a_1, a_2) = 0.56$



(b) Representation on the 1^{st} layer: $r(aw_{11}, aw_{12}) = 0.72$



(c) Representation on the 2^{nd} layer: $r(aw_{21}, aw_{22}) = 0.93$



(d) Representation on the output layer: $r(out_1, out_2) = 1.0$

Figure 3.9: Data representation evolution on different layers of DNN for linearly non-separable two class problem

Noticeably, the correlation grows and the weighted original data distribution becomes closer to linearly separable space. Also, one can see an increase of r, mean-

ing that data become less random with the growth of hidden layers in the network. Therefore, we believe that such an intrinsic DNN property can help to tackle NF limitations with respect to multinomial classification.

As was shown in the Figure 3.9, each new layer of DNN adds non-linearity by making non-linearly separable classes align towards a separable hyperplane.

3.3.1 Deep Mapping of Feature Space

The main challenge with shallow single-layer neural architectures when dealing with multinomial data is the inability to differentiate samples from different classes because of similar properties. It has been demonstrated [377] that the overall mean absolute PCC $|\bar{r}| \rightarrow 0.0$ of such data, meaning that it becomes a rather random distribution without any specific regions with correlations that can be specifically approximated. In such cases, NF results in a few general rules, defined by S_P in Equation 3.4, giving a lower classification accuracy. To overcome this limitation, it is necessary to increase $|\bar{r}| > 0.0$ by mapping the input characteristics of the given malware to another hyperspace with less uncertainty. The general representation of the next hidden ANN layer is defined as an activation function over a linear combination: $h_i^j = g(\sum_{i=0}^{M-1} w_i^j \cdot h_i^{j-1})$ of $j-th$ layer and $i-th$ hidden unit using differentiable activation function $g(h_i^j) = 1/(1+e^{-h_i^j})$. The total error function $E(W)$ of the output layer of DNN can be expanded as follows for the desired output class d, as depicted in the Equation 3.26.

$$
\begin{aligned}
E(W) = \frac{1}{2} \cdot [d - \frac{1}{1+e^{-h_i^j}}]^2 = \\
\frac{1}{2} \cdot [d - g(\sum_{i=0}^{M-1} w_i^j \cdot \frac{1}{1+e^{-h_i^j}})]^2 = \cdots = \\
\frac{1}{2} \cdot [d - g(\sum_{i=0}^{M-1} w_i^j \cdot g(\dots w_i^1 \cdot g(\sum_{i=0}^{M-1} w_i^0 \cdot x_i) \dots))]^2
\end{aligned}
\tag{3.26}
$$

From the expanded equation above, we can see that the set of deepest weights of the input layer $\bar{w^0}$ have the least influence on the derivative $\frac{d\,E(W)}{dW}$, yet the most on weights adjustments used for feature mapping. In fact, the deltas $\delta(\bar{w^j}) >> \delta(\bar{w^{j-1}})$ for the sample output deltas $\epsilon(E(\bar{w^j})) \approx \epsilon(E(\bar{w^{j-1}}))$. The higher layers require less adjustment of the weights to cause the same function growth. So, we can posit ideally $\lim_{L\rightarrow\infty} E(W) = 0$ for the number of hidden layers L, which is achieved by mapping the feature space to a distribution with $|\bar{r}| > 0$.

3.3.2 Integration with the 1^{st} Stage of Neuro-Fuzzy

Keeping in mind the influence of each hidden layer's weights decreasing from input to output, the decrease in uncertainty takes the biggest leap in the first DNN layers. To incorporate it into the NF structure, we $X' = \Gamma(X)$, where the mapping function is based on the non-linear transformation from DNN for desired depth measure Q:

$$\Gamma = g(\sum_{i=0}^{M-1} w_i^Q \cdot h_i^Q) \tag{3.27}$$

Independently from Q, such a method gives the freedom to increase δ in the Equation 3.4, also reducing the uncertainty of the data. Unlike Zhou et al. [462], this method allows for integration in the two-stages method proposed by Kosko [233], resulting in a better configuration of fuzzy patches after the unsupervised 1^{st} stage of NF learned by SOM, and an increased precision of the corresponding MF on the 2^{nd}:

$$\mu_j(X') = e^{-\frac{1}{2}(\Gamma(X)-\Gamma(C))^T \ \Sigma^{-1} \ (\Gamma(X)-\Gamma(C))} \tag{3.28}$$

Through the suggested modification, we add a new level of abstraction to the conventional NF approach, making it work with even more uncertain data, as shown in the Figure 3.10.

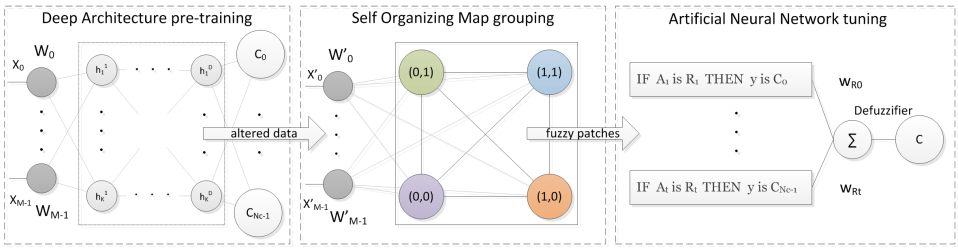


Figure 3.10: Proposed Deep Neuro-Fuzzy approach based on classic two stages approach accroding to Kosko [233]

This NF architecture consists of the following processing stages: (i) *pre-training*, which uses multilayer perception; (ii) 1^{st} stage, consisting of unsupervised fuzzy patches extraction from the pre-trained data; and (iii) 2^{nd} stage, consisting of supervised tuning of the fuzzy rules model by adjusting corresponding rules weights.

3.4 A New Method of On-line MLP Training Using Genetic Algorithm[4]

Despite the successful application of EBP, there are multiple explanations as to why this method fails to learn properly from the given data. (i) There may be several occurrences of local minima. (ii) The wrong placement of a pre-defined set of initial parameters such as learning rate α will mislead the optimization procedure. Thus, one can highlight the following difficulties related to the usage of EBP: convergence is not guaranteed, it can be slow, and it depends on the input data parameters. (iii) The challenge of training an on-line model from the data within as low a number of iterations as is possible is apparent. From the literature, we can see that EC has been applied for stochastic optimization in MLP before to reduce convergence time. It is also vital that the decision is made in a short period of time, because otherwise it can cause privacy breaches due to unreasonable delays in re-training an AC mechanism. In contrast to previous works [122, 208, 213, 347, 419], we suggest the application of an optimization procedure for individual α.

3.4.1 Single-step On-line Learning of MLP

The aforementioned optimization problem in single-step online MLP learning is caused by non-linearity and a high level of abstraction.

Lemma: *The error function $E(W)$ in a single-step MLP is non-monotonic with multiple extreme points due to several layers of non-linearity introduced by nested hidden layers. As a result, conventional GD-based optimization methods may fail to find a global optimal set of weights W.*

The learning rate α in weights adjustments shall not be constant on every iteration; this will result in a faster decrease of the $E(W)$ on each learning iteration. Furthermore, each of these adjustment steps consists of an additional unconstrained optimization procedure that is aimed at finding a corresponding optimal α by means of a meta-heuristic real-valued GA. The GA enables multiple hypothesis evaluations at the same time since the problem is to find an appropriate value of α. This is done towards hardening the robustness against non-deterministic patterns in the access sequence. We consider MLP, which is based on a differentiable sigmoid activation function [232]. It can be seen that the error function $E(W)$ is a non-linear one that includes the recursive additive composition of the neurons activation functions for

[4]The ideas of this section are published under the contribution [368]

each given labelled dataset:

$$E(W) = \frac{1}{2} \cdot (d - y)^2 = \frac{1}{2} \cdot (d - \frac{1}{1 + e^{-h_i^j}})^2, \tag{3.29}$$

Remark. The complexity of the function $E(W)$ will grow with the number of hidden layers. Thus, the function will have a corresponding recursive form for each of the given labelled data samples in a single-step approach as shown in the Eq. 3.30.

$$\begin{aligned} E(W) = \frac{1}{2} \cdot [d - \frac{1}{1 + e^{-h_i^j}}]^2 = \\ \frac{1}{2} \cdot [d - g(\sum_{i=0}^{M-1} w_i^j \cdot \frac{1}{1 + e^{-h_i^j}})]^2 = \cdots = \\ = \frac{1}{2} \cdot [d - g(\sum_{i=0}^{M-1} w_i^j \cdot g(\dots w_i^{j0} \cdot g(\sum_{i=0}^{M-1} w_i^0 \cdot x_i) \dots))]^2 \end{aligned} \tag{3.30}$$

The function in Eq. 3.30 represents an error surface $E(W)$ that has a non-linear dependency with respect to a set of neuron weights W. From this perspective, the influence of the weights in the initial layer will have a greater degree of non-linearity than the next layer. Rojas [336][Chapter 7] studied a similar example of the error function and depicted a possible local minima that affects the optimization. Therefore, there are multiple challenges to conventional GD optimization, possibly resulting in local optima solutions [122]. Multiple plateaus and local minima make it unlikely to achieve the global minima, so GD will be stuck sub-optimally during the weights update process. In fact, for an arbitrary weight w_i^j of the layer j, the next function's $E(W)$ limit will have a place as shown in the Eq. 3.31, considering thw MLP error function described in the E. 3.30. This is based on the limit's property of the composite continuous functions $\lim f[g(x)] = f[\lim g(x)]$ according to Stein [401], since the sigmoid function $g(h_i^j)$ is a continuous and differentiable function. Moreover, the neighbourhood of the global minima of the derived limit is inside the tolerance interval $L \pm \epsilon$ for the given neighbourhood of the optimal value of the weight $w_i^{j^{optimal}} \pm \delta$, as studied by Exner [139] while continuously changing the input data sample X. Also, we can see that the nested combination of multiple monotonically increasing sigmoid functions will result in a number of local optima, rather than one global one. Such combinations of the sigmoid function will give a complex non-linear high level abstraction for the same input data pattern, and different weight values. Thus, it is nearly impossible to define the exact value of the limit for the global extreme point neighbourhood since the resulting limit is a complex one, and the vector of the weights needs to

be closer to the global optima. The limit is not a constant value however, as there is a constant concept drift at each iteration in data stream mining. As a result, the neighbourhood of the global optima $w_i^{j\,optimal}$ is changing stochastically with each new data sample, which means that the error function $E(W)$ has an inconsistent global optima region, making the usage of the fixed-α method less efficient.

$$\begin{aligned} \lim_{w_i^j -> w_{optimal}} E(W)|_{X=constant} = \\ \lim_{w_i^j -> w_{i\,optimal}^j} \frac{1}{2} \cdot [d - g(\sum_{j=0}^{M-1} w_i^j \cdot g(\dots w_i^0 \cdot \\ g(\sum_{i=0}^{M-1} w_i^0 \cdot x_i) \dots))]^2 \approx \\ \approx \frac{1}{2} \cdot [g(\dots \lim_{w_i^j -> w_{i\,optimal}^j} g(\sum_{i=0}^{M-1} w_i^0 \cdot x_i) \dots) \dots]^2 \\ \in [L \pm \epsilon] \end{aligned} \tag{3.31}$$

There is a need to apply more advanced techniques for the weights optimization rather than a conventional one like constant or increasing learning rate α. $E(W)$ is located within some ϵ of the L, meaning that for the different input X, the global optima will be different. For our purpose, we use a sigmoid activation function for each neuron since it is differentiable [283], and most suitable for learning in the case of two-class problems: either (*"denied"* or *"allowed"*). The main point of optimization is to find an optimal set of weights W that give the lowest possible value of the error function $E(W)$ during single-step learning. Therefore, it is unacceptable to apply purely unimodal optimization and line searching, because they are exposed to premature convergence according to Salomon [351]. Unimodal heuristic optimization has monotonicity as its necessary criteria according to Doerr et al. [182], yet this is not achieved. As result, the found solution will be a local one.

3.4.2 An Optimal Individual Learning Rate α Prediction Using Genetic Algorithm

Our method targets the usage of individual optimal α for each of the weights on each layer that make MLP converge faster than conventional unified fixed-α or deterministic decreasing/increasing α approaches. Additionally, the single-step on-line learning is applied since the availability of data samples for training is limited. For on-line incremental learning, the α has to be optimal on each step for eliminating the accumulation of the error residuals. So, meta-heuristic EC tends

to solve the problem more reliably and quickly when classical unimodal search methods are slow.

Proposal: *Evolutionary Computing, Genetic Algorithm in particular, is applied as one-dimension optimization in a single-step MLP learning to facilitate a proper individual α-determination for each particular weight w_i^j optimal update.*

Non-monotonic functions such as $E(W)$ have lower chances to be optimized due to multiple extreme points; we therefore consider EC methods as the most promising for such tasks. MLP provides nested optimization problem. The *primary* problem includes seeking optimal weights as shown in the Figure 3.32, which gives a value of the error function referring to the Equation 2.36.

$$min_{W \in \mathbb{R}^M} \; E[W - \alpha \cdot \nabla E(W)] \tag{3.32}$$

The *secondary* optimization problem is to find a set of optimal steps α as shown in the Figure 3.33, which will result in a global optimal solution for single-step algorithms to avoid long iterative learning. However, at this point, we need to consider each individual α_{ij} that employs mutation and crossover operations to be optimized by GA, which increases the chance to cover as much search space as possible while keeping the weights constant.

$$min_{\bar{\alpha} \in \mathbb{R}^M} \; E[W - D(\bar{\alpha}) \cdot \nabla E(W)] \mid_{W=const} \tag{3.33}$$

where D - is a diagonal matrix.

Remarks. The surface of the error function has a non-linear dependency on the weights values, which means that the weights update process has to be performed by meta-heuristic optimization methods in order to avoid premature convergence. Conventionally, GSS or similar line search methods are applied as a unimodal optimization. But it does not work well without reliable information about the borders of the optimal learning rate, and is usually very slow. Naturally, GA will make it converge faster.

Proposed methodology & Algorithm

Under the aforementioned constraints in on-line incremental learning, we propose to use single-step MLP, solving the nested optimization problem to find an optimal set of individual α for the weights update. As a result, the weights are only corrected based on the optimal learning rate. Additionally, it is important for privacy protection that its application maintains the trade-off between speed and reliability of the answer. The algorithm of the proposed method is defined in the Listing 3. The conventional Error Back Propagation method was modified accordingly. For

this algorithm, we make the assumption that each hidden layer has the same dimensionality as the input data vector. The computational complexity will be as follows: for each neuron's weight, there will be $pop\_size(\cdot p_{crossover} + p_{mutation}) \cdot N_{epochs}$ recalculation of the fitness function in a case-by-worth scenario. However, this can be decreased by introducing the memory of the fitness function values. Additionally, there is a chance of getting to the sub-optimal or optimal $\alpha_{optimal}$ within a fewer amount of epochs in comparison to the Brent's Method (Golden Section Search) or Fibonacci method, since they have rather linear convergence according to Press [321].

The real-valued does not require additional binary mapping schemes, and therefore the α values are used as chromosomes and the error function $E(W)$ is defined as a fitness one. The corresponding *Arithmetic Crossover* was performed for the crossover operation as discussed in the paper by Köksoy et al. [237]. Additionally, the mutation was done as a real-valued random uniform mutation of a chosen chromosome as shown by Adewuya [43]. Here, we concentrated on deriving as close to an optimal learning rate α as possible in a short time frame. Information Security tasks often put speed and response time constraints on hard computational tasks rather than constraints on answer precision.

3.5 Analysis of Complexity of Novel Neuro-Fuzzy[5]

In this Section, we analyse the complexity of the improved Neuro-Fuzzy in comparison to the simple method and the method proposed by Kosko. This is one of the most important considerations when dealing with large-scale data analytics. It will be shown further that the proposed method has a lower number of basic computational operations.

3.5.1 Algorithm of the Proposed Novel Neuro-Fuzzy Method

The full description of the proposed procedures for NF using SOM for large-scale datasets with an emphasis on interpretability is given in the Algorithm 4. It is divided into four distinct functions: Line 1 describes the assignment of initial parameters necessary for the proper operation of the algorithm where α_{χ^2} is a pseudo-radius used in the elliptic fuzzy patches calculation, $SOMepochs$ describes the number of epochs SOM is trained on the 1^{st} stage of NF, $NFepochs$ - the number of iterations ANN is trained on the 2^{nd} NF stage, and BA is a fraction of the data samples from the initial dataset used in bootstrap aggregation.

The function on Line 2, $OptimalSize$, estimates the optimal size of SOM according to the properties of given data $\bar{x}$, which are described in the subsection

[5]The main ideas of this section are published under the contributions [376]

Algorithm 3 Optimization of α-rate in single-step MLP training using real-valued GA

$x \leftarrow x_{new}$
$\overline{\overline{w}} \leftarrow \overline{\overline{w_{previous}}}$
$\delta_{output} = g(h) \cdot (d - y) \cdot (1 - g(h))$
for all hidden layer **do**
  for all neurons in a current hidden layer **do**
    $\delta_i^j \leftarrow g(h) \cdot (1 - g(h)) \cdot w_i^j \cdot \delta_{output}$
    initialization(α_{random},pop_size);
    while $N_{epochs} < N_{epochsMax}$ or $|Fit_k - Fit_{k-1}| < \epsilon$ or $|Fit^{'}| < \epsilon$ **do**
      MUTATION($p_{mutation}$,α)
      CROSSOVER($p_{crossover}$,$\alpha 1$,$\alpha 2$)
      SELECTION(α);
      $\alpha_{optimal} \leftarrow \alpha_{selected}$
    end while
    $w_{ij} \leftarrow w_i^j + \alpha_{optimal} \cdot \delta_i^j \cdot g(h_i^j)$
  end for
end for
return $\overline{\overline{w}}$
function MUTATION($prob_{mutation}$,α)
  $d \leftarrow random(0, 1)$ (generation of a real number)
  $\alpha_{mutated} \leftarrow \alpha_{mutate} \pm d$
  return $\alpha_{mutated}$
end function
function CROSSOVER($prob_{crossover}$,$\alpha 1$,$\alpha 2$)
  $d \leftarrow random(0, 1)$ (generation of a real number)
  $offspring1 \leftarrow d \cdot y_i + (1 - d) \cdot x_i$
  $offspring2 \leftarrow d \cdot x_i + (1 - d) \cdot y_i$
  return $\alpha_{off1}, \alpha_{off2}$
end function
function SELECTION(α)
  $Fit \leftarrow E(W)|_\alpha$
  return $\alpha_{optimal}$
end function

3.1, Proposal 2. As a result, the optimal weight and height of the SOM grid are derived. The function invoked on Line 3, $SOMtraining$, describes conventional iterative training of SOM using BMU based on previously defined parameters. The outcome of such training is a set of clusters for each of the classes, i.e. a set of data samples at each BMU. The third function on Line 4 $clustersToRules$ converts clustered data into corresponding fuzzy patches parameters which are used later to form fuzzy rules. The description of this process is justified in the subsection 3.2, Theorem 1. Finally, the function on Line 5 $fuzzyRules$ covers the process of fuzzy rules tuning using conventional ANN training as described by Kononenko et al. [232]. It means that weighted input data (fuzzy rules) are fed to a combination function that is used together with the activation function during the iterative training (modification of weights) $NFepochs$ times. The used MF $ProposedMF$ is described in the subsection 3.2, Theorem 2. Finally, the fuzzy inference is based on fuzzy rules with the corresponding MF and weights from ANN tuning.

3.5.2 Complexity Evaluation

One of the main concerns when dealing with data analysis in Digital Forensics is the delay required to build a classification model and label the questioned network packet. Here, the analysis of the computation complexity is presented with respect to the approximate number of operations required to perform on the input data as discussed by Arora et al. [62]. We consider the tentative computation cost (big O notation as time complexity) of each feature a_i processing in the vector of the input data. NF consists of two training stages: grouping by SOM and the corresponding fuzzy rules tuning by ANN. Later, we also consider inference complexity.

Training Complexity

It defines the amount of computations required for a method to train from the given dataset to get a complete classification model. We consider a worthy case scenario that includes calculations without empty nodes in SOM clustering.

1. *Vesanto as SOM size determination and simple rectangular patches with triangular MF*. On the 1^{st} stage of NF, we need to perform the following steps:

   (a) *SOM size estimation* is done based on eigendecomposition on the whole original dataset, as described by Smith [396]:

   $$N_S \cdot {N_F}^3 \tag{3.34}$$

   (b) *SOM training* includes iterative training using Best Matching Unit (BMU) based on the distance calculation according to Maiorana et al. [260].

Algorithm 4 Proposed modifications of Neuro-Fuzzy method

$\alpha_{\chi^2} \leftarrow 95\%$; $SOMepochs \leftarrow 100$; $NFepochs \leftarrow 10$; $\bar{x} \leftarrow data$; $BA \leftarrow 1\%$
$Hgrid, Wgrid \leftarrow OptimalSize(\bar{x})$
$clustersConfig \leftarrow SOMtraining(\bar{x}, Hgrid, Wgrid, BA, SOMepochs)$
$fuzzyPatchesParameters \leftarrow clustersToRules(clustersConfig)$
$fuzzyRules \leftarrow trainingANN(\bar{x}, fuzzyPatchesParameter, NFepochs)$
function OPTIMALSIZE($\bar{x}$)
    $S_{min} = 2^2$; $S_{max} = 5^2$; $nC \leftarrow 2$
    $Corr(\bar{x}) = PearsonCorrelationCoeffMatrix(\bar{x})$
    $E0, E1 \leftarrow Eigendecomposition(Corr(\bar{x}))$; $alpha \leftarrow \frac{E0}{E1} \cdot avg(Corr(\bar{x}))$
    $S_{opt} \leftarrow S_{min} + (S_{max} - S_{min}) \cdot \alpha$
    $Hgrid = \lceil \sqrt{S_{opt}} \rceil$; $Wgrid = \left\lfloor \frac{S_{opt}}{Hgrid} \right\rfloor$
    return $Hgrid, Wgrid$
end function
function CLUSTERSTORULES($clustersConfig$)
    for all clusters extracted from SOM **do**
        $Cov(\bar{x}) \leftarrow CovarianceMatrix(cluster)$
        $Cov^{-1}(cluster) \leftarrow matrixInverse(Cov(cluster))$
        $FuzzyPatches_{\mu} = centroids(cluster)$; $FuzzyPatches_{\sum} = (Cov^{-1})$
    end for
    return $FuzzyPatches$
end function
function PROPOSEDMF($\alpha_{\chi^2}, sample$)
    $mahalanobisD \leftarrow (sample - \mu)^T \cdot \sum^{-1} \cdot (sample - \mu)$
    $\chi^2 \leftarrow contingencytable(DF)|_{\alpha_{\chi^2}}$
    if $mahalanobisD \geq \chi^2$ **then**
        $mf \leftarrow e^{-\frac{1}{2} \cdot \chi^2}$
    else
        $mf \leftarrow e^{-\frac{1}{2} \cdot mahalanobisD^2}$
    end if
    return mf
end function

Later comes the assignment of the samples for a particular BMU node with the corresponding weights update:

$$N_S \cdot (S_V \cdot N_S + S_V \cdot log(S_V) + S_V \cdot N_F) \tag{3.35}$$

where S_V denotes the number of SOM nodes suggested by the Vesanto method.

(c) *Fuzzy rules parameters estimation* includes calculation of 1^{st} and n^{th} order statistics using min and max values of the elements per feature. In such a way, the centers and bases of the triangular MF are found for each of the rules:

$$N_{R\,V} \cdot N_F \cdot \frac{N_S}{N_{R\,V}} = N_S \cdot N_F \tag{3.36}$$

where $N_{R\,V}$ is a number of rules extracted after SOM training using a SOM size defined by Vesanto method.

On the 2^{nd} step of NF, training ANN is used to tune each particular rule according to the full original dataset:

$$N_e \cdot N_S \cdot N_{R\,V} \cdot {N_F}^2 \tag{3.37}$$

where N_e denotes the number of epochs used in ANN.

2. *Vesanto as SOM size and Kosko method with shadow-based triangular MF from the circumscribed rectangular over the hyperellipsoid.* In the 1^{st} stage of NF, we need to perform following steps:

   (a) *SOM size estimation* is the same as above.

   (b) *SOM training* is the same as above.

   (c) *Fuzzy rules parameters estimation* is more cumbersome than the previous step, and was suggested by Kosko [233]. It includes a zero-mean matrix calculation for each of the SOM clusters, and covariance and inverse covariance matrices calculation. This is finished with the eigendecomposition of the inverse covariance matrix and calculation of the corresponding triangular MF parameters for each of the rules:

$$N_{R\,V} \cdot (\frac{N_S}{N_{R\,V}} \cdot N_F + \frac{N_S}{N_{R\,V}} \cdot {N_F}^3 + \tag{3.38}$$

$$+\frac{N_S}{N_{R\,V}} \cdot {N_F}^3 \frac{N_S}{N_{R\,V}} \cdot {N_F}^3 + \frac{N_S}{N_{R\,V}} \cdot {N_F}^2) = \tag{3.39}$$

$$N_S \cdot (3 \cdot {N_F}^3 + {N_F}^2 + N_F) \tag{3.40}$$

where $N_{R\,V}$ is a number of rules extracted after SOM training using SOM size, defined by the Vesanto method.

The 2^{nd} sage is identical to the one described above.

3. *Proposed improved method with bootstrap aggregation*. The 1^{st} step differs significantly from the previous methods:

   (a) *SOM size estimation* is done according to the suggested method based on the PCC:
   $$N_S \cdot {N_F}^3 \cdot N_F \tag{3.41}$$

   (b) *SOM training* is basically similar to the previous simple and Kosko methods, yet we used a considerably low number of samples D_l in order to bootstrap aggregation and a new optimal SOM size metric S_P:
   $$D_l \cdot (S_P \cdot D_l + S_P \cdot log(S_P) + S_P \cdot N_F) \tag{3.42}$$

   (c) *Fuzzy rules parameters estimation* similar to the method used by Kosko, yet we do not need to perform eigendecomposition to extract eigenvalues and vectors for MF construction. Instead, we use an inverse covariance matrix as a part of MF:
   $$N_{R\,P} \cdot (\frac{D_l}{N_{R\,P}} \cdot N_F + \frac{D_l}{N_{R\,P}} \cdot {N_F}^3 + \tag{3.43}$$
   $$+\frac{D_l}{N_{R\,P}} \cdot {N_F}^3 + \frac{D_l}{N_{R\,P}} \cdot {N_F}^2) = \tag{3.44}$$
   $$D_l \cdot (2 \cdot {N_F}^3 + {N_F}^2 + N_F) \tag{3.45}$$

   where $N_{R\,P}$ is a number of rules extracted with the help of proposed optimal SOM size.

   On the 2^{nd} step of the NF stage, sequence matrix multiplications and transposition are required, so the complexity will be as follows:
   $$N_e \cdot N_S \cdot N_{R\,P} \cdot {N_F}^6 \tag{3.46}$$

To summarize, we compare complexities of both NF stages in the Table 3.3. In the table below, we target large-scale analysis that gives $S_V >> S_P$ and $N_S >> N_F$.

The proposed method significantly reduces the complexity of the 1^{st} stage of Neuro-Fuzzy $N_{R\,V} >> N_{R\,P}$ Basically, we can see that the 2^{nd} stage of NF can take a lot of time, and optimization is required. In particular, we can reduce

Table 3.3: Analysis of computation complexity of three NF methods: S is for simple, K is for Kosko, and P is for proposed

NF	Number of operations for training	Approximation
S	$N_S \cdot {N_F}^3 + N_S \cdot (S_V \cdot N_S + S_V \cdot log(S_V) + S_V \cdot N_F) + N_S \cdot N_F + N_e \cdot N_S \cdot N_{R\,V} \cdot {N_F}^2$	$\approx {N_S}^2 \cdot S_V + N_e \cdot N_S \cdot N_{R\,V} \cdot {N_F}^2$
K	$N_S \cdot {N_F}^3 + N \cdot (S_V \cdot N_S + S_V \cdot log(S_V) + S_V \cdot N_F) + N_S \cdot (3 \cdot {N_F}^3 + {N_F}^2 + N_F) + N_e \cdot N_S \cdot N_{R\,V} \cdot {N_F}^2$	$\approx N_S^2 \cdot S_V + N_e \cdot N_S \cdot N_{R\,V} \cdot {N_F}^2$
P	$N_S \cdot {N_F}^4 + D_l \cdot (S_P \cdot D_l + S_P \cdot log(S_P) + S_P \cdot N_F) + D_l \cdot (2 \cdot {N_F}^3 + {N_F}^2 + N_F) + N_e \cdot N_S \cdot N_{R\,P} \cdot {N_F}^6$	$\approx N_S \cdot {N_F}^4 + N_e \cdot N_S \cdot N_{R\,P} \cdot {N_F}^6$

the number of trainings by applying parallel optimization for rules training that will result in the following complexity on a p-threaded platform:

$$N_e \cdot N_S \cdot N_{R\,P} \cdot {N_F}^6 \rightarrow N_e \cdot N_S \cdot \frac{N_{R\,P}}{p} \cdot {N_F}^6 \tag{3.47}$$

Inference Complexity

Inference Complexity denotes the amount of computations required to calculate the membership value of an unknown data sample.

1. *Triangular MF* was used for the first two methods and needed ${N_F}^2$ multiplication and division to calculate the membership value of the rule.

2. *Modified Gaussian MF* proposed in this paper required ${N_F}^6$ operations due to the multiplication of the transposed zero-mean matrix, inverse covariance, and zero-mean matrix again. This is why the proposed MF may take to time to calculate the membership value. It can be improved with the help of parallel optimization.

Chapter 4

Application in Digital Forensics Science

Digital Forensics includes many specific domains related to Cyber Crime Investigation. In order to show the successful application of Machine Learning, the Neuro-Fuzzy rule-extraction classification method in particular, we have studied a number of practical applications reflected in the sections below. The following areas were analysed: Windows Malware detection and analysis, Network Intrusion Detection, Web Application Firewall, Network Forensics Readiness and Mobile-Device virus analysis.

The experimental part was conducted at NTNU Digital Forensic group during 2013-2017 and includes comprehensive evaluation of the proposed methods with respect to challenges and requirements in variety of different real-world applications: (i) Using state of the art datasets like the Android malware dataset, obsolete KDD CUP 1999, and PKDD 2007 dataset, we performed a comprehensive evaluation of the applicability of the proposed improvements. (ii) A novel large-scale collection of the Portable Executable 32bit malware files was composed as a part of this PhD thesis. It consists of 328,000 labelled malware samples that represent 10,362 families and 35 categories, which was further tested as a non-trivial multinomial classification problem, and was neither sufficiently studied in the literature nor explored before. In addition to this, we successfully demonstrated the advantage of the proposed method using large-scale datasets such as SUSY and HIGGS.

4.1 ML-aided Windows Malware Detection[1]

Users of MS Windows Operation Systems have been under attack since it was released toward the end of the 1990's due to the numerous versions and inherent security flaws, also dramatically enhanced by a lack of user awareness. The file format used by the operating system is Portable Executable (PE), developed for executable and dynamic-link libraries[302]. Below, first we present a study of the in-depth static code analysis of PE files to understand malicious patterns without file execution. Second, we display the investigation of the applicability of certain Machine Learning techniques. As a result of this investigation, a taxonomy of Machine Learning and feature selection methods are proposed. Later, a practical tutorial is given. This is supported by extensive experiments done in order to get a clear picture on the malware detection performance using a baseline dataset for training.

4.1.1 Datasets

As for MALWARE PE32 COLLECTION, one can see that the number of viruses developed for Windows is large, therefore, it is hardly possible to completely catch up with new samples discovered in the wild on a daily basis. Though our goal was to demonstrate practical aspects of PE analysis, we tried to get larger sets to be able to achieve better coverage in the software. The original aim was to gather all possible combinations of viruses, so there was a need for intensive preprocessing to extract the target group of PE32 executables. Our main malware source is an archive *VirusShare repository* accessible through VirusShare tracker [19]. We decided to consider the two following archives: *VirusShare_00000.zip* created on 2012-06-15 00:39:38 with a size of 13.56 GB and *VirusShare_00207.zip* created on 2015-12-16 22:56:17 with a size of 13.91 GB. Both archives contain 131,072 uncategorised and unsorted samples gathered by a unique md5 sum. They will be referred to further as **malware_000** and **malware_207**.

To the authors knowledge, there have not been published any large BENIGN SOFTWARE REFERENCE DATASETS, so we had to create our own set of benign files. Even though the National Software Reference Library [300] exists, only MD5 sums are available to public. Since the focus of the paper is mainly on PE32 for Windows NT OS families, we decided to extract corresponding known-to-be-good files from different versions of MS Windows, including different software and multimedia programs installations that are available for this OS to get better coverage. To perform this, we used the previously mentioned VDS with corresponding installation of various OS inside to speed up the process of analysis. The versions

[1]The main ideas of this section are published under the contribution [382]

that we processed are: Windows XP 32-bit (with MS office and other programs installed), Windows 7 32 bit, Windows 8 32bit and Windows 10 32bit installation. This will be referred to further as **benign** dataset. To be able to perform experiments on the dataset, we had to filter out irrelevant samples, which are out of the scope of this paper.

After collecting all possible files and performing the pre-processing phase, we ended up with the following sets as presented in the Table 4.1. Even after performing extensive malware collection from different OS in the Windows NT family, the amount of files is lower due to the similar binaries used across different versions of MS Windows.

Table 4.1: Characteristics of the dataset collected and used for our experiments after filtering PE files

Dataset	Number of files	Size
Benign	16,632	7.4GB
Malware_000	58,023	14.0GB
Malware_207	41,899	16.0GB

After performing the literature study we evaluated how well the collected data describes the current distribution of malware samples with respect to different versions of Windows NT OS. Moreover, the accuracy of ML methods was studied using a set of extracted features.

4.1.2 Experimental Setup

For the experiments, we used a set of benign and malicious samples. However, the processing of 100k samples imposed limitations and required non-trivial approaches to handle such a large amount of files. We discovered that common ways of working with files in directories such as simple *ls* and *mv* in *bash* take an unreasonable amount of time to execute. Also, there is no way to distinguish files by extension like **.dll* or **.exe* since the names are just *md5* sums.

We characterized the executables by means of architecture (PE32 32bit), on which it intended to run like:

```
PE32 executable (GUI) Intel 80386, for MS Windows
PE32 executable (DLL) (GUI) Intel 80386, for MS
    Windows
PE32 executable (GUI) Intel 80386, for MS Windows, UPX
     compressed
```

Following our purpose to concentrate only on 32bit architecture, only pure PE32 are filtered out from all possible variants of PE32 files.

After filtering all benign and malicious PE32 files, multiple rounds of feature extraction were performed according to methods used in the literature. Finally, we insert the extracted features into the corresponding MySQL database to ease the handling, feature selection, and machine learning processes.

4.1.3 Results & Analysis

This part presents the results of applying Naive Bayes, BayesNet, C4.5, k-NN, SVM, ANN, and NF machine learning algorithms against static features of our dataset namely PE32 header, Bytes n-gram, Opcode n-gram, and API calls n-gram.

1. **PE32 header**

   PE32 header is one of the most popular sources of binary files characteristics used in static analysis for malware detection. The most important thing that we discovered and that can be relevant to threat intelligence is the significance of the features extracted from PE32 headers. We performed feature selection using *Cfs* and *InfoGain* methods while using 5-fold cross-validation; results are presented in the Figure 4.2. The *InfoGain* method is based on the ranking and estimation attribute's merit. As a result, one can see the degree to which each of the attributes are relevant to malware classification problems. From the other side, *Cfs* gives us only a sub-set of features that achieve higher accuracy considering their correlation.

   We can clearly see that the features from the *Short Info* section in PE32 headers can be used as stand-alone indicators of malware, including different epochs. The number of directories in this section, as well as the file size and flag of EXE or DLL, have bigger merits in comparison to other features. To contrast, *Anti Debug* and *Suspicious API* sections from *PEframe* do not necessarily indicate whether a binary file belongs to malware or goodware. Finally, we can say that the digital signature and *Anti VM* files in PE32 headers are almost irrelevant for the malware detection task. *PEframe* was mentioned by SANS [457] as one of the prominent tools to analyse suspicious executables on Windows, since it is able to detect relevant information earlier, in addition to the standard fields in PE headers. Hahn made a contribution of a robust static analysis of portable executables for malware detection [173]. The work includes possible malformations of the PE format that can be used to identify malicious intentions. The authors also stated that PE format has an ambiguous documentation that does not necessarily get used on purpose. Since the number of fields is large, it requires some knowledge

Table 4.2: Feature selection on PE32 features. Bold font denotes selected features according to *InfoGain* method

Benign vs Malware_000		Benign vs Malware_207		Malware_000 vs Malware_207	
Information Gain					
merit	attribute	merit	attribute	merit	attribute
0.377	**ShortInfo_Directories**	0.369	**ShortInfo_DLL**	0.131	**ShortInfo_FileSize**
0.278	**ShortInfo_DLL**	0.252	**ShortInfo_Directories**	0.094	ShortInfo_Detected
0.118	**AntiDebug**	0.142	**ShortInfo_FileSize**	0.064	SuspiciousAPI
0.099	Packer	0.105	**SuspiciousSections**	0.044	ShortInfo_Directories
0.088	SuspiciousSections	0.101	**SuspiciousAPI**	0.036	Packer
0.082	ShortInfo_Xor	0.089	AntiDebug	0.028	AntiDebug
0.076	SuspiciousAPI	0.084	ShortInfo_Detected	0.017	SuspiciousSections
0.045	ShortInfo_FileSize	0.054	ShortInfo_Xor	0.016	Url
0.034	ShortInfo_Detected	0.050	Packer	0.015	AntiVM
0.022	Url	0.036	Url	0.012	ShortInfo_Xor
0.004	AntiVM	0.002	AntiVM	0.002	ShortInfo_DLL
0	ShortInfo_Sections	0	ShortInfo_Sections	0	ShortInfo_Sections
0	DigitalSignature	0	DigitalSignature	0	DigitalSignature
Cfs					
attribute		attribute		attribute	
ShortInfo_Directories		ShortInfo_Directories		ShortInfo_Directories	
ShortInfo_Xor		ShortInfo_Xor		ShortInfo_FileSize	
ShortInfo_DLL		ShortInfo_DLL		ShortInfo_Detected	
ShortInfo_Detected				Packer	
Url					

to identify key indicators. Therefore, we believe that an automated analysis may help to identify key compromise indicators to help malware analysts. Moreover, there can be pieces of information describing unusual findings in data directories or a windows specific field. Furthermore, we performed the exploration of selected ML methods that can be used with selected features. By extracting the corresponding numerical features mentioned earlier, we were able to achieve the classification accuracy presented in the Table 4.3. It can be seen that such static analysis methods give high accuracy on most ML methods. Moreover, we are able to differentiate between different epochs of malware using these features. The Table 4.2 also presents the accuracy of the ML method after performing feature selection. Here, we used whole feature sub-sets defined by the *Cfs* method and features with merit ≥ 0.1 detected by *InfoGain*.

Malware vs goodware datasets can be easily classified using the full set as well as a sub-set of the features. One can also notice that ANN and C4.5 performs much better that other methods.

It can also be seen that the quality of these features can be considered as

Table 4.3: Comparative pair-wise binary classification accuracy between benign, malware_000 and malware_207 datasets based on features from PE32 header, in %.

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
All features							
Bn vs Ml_000	90.29	91.42	**97.63**	97.30	87.75	95.08	92.46
Bn vs Ml_207	88.27	91.21	**96.43**	95.99	84.88	93.24	89.03
Ml_000 vs Ml_207	63.41	71.59	**82.45**	82.11	73.77	69.99	69.01
Information Gain							
Bn vs Ml_000	88.32	89.17	**94.09**	94.01	94.09	93.51	87.53
Bn vs Ml_207	87.25	90.39	**95.06**	94.58	84.55	92.37	87.88
Ml_000 vs Ml_207	58.26	67.05	67.77	**70.70**	69.46	63.19	51.31
Cfs							
Bn vs Ml_000	89.35	90.89	**95.39**	95.38	95.16	93.69	85.85
Bn vs Ml_207	86.88	89.67	91.61	**91.68**	91.68	91.68	81.91
Ml_000 vs Ml_207	67.45	70.95	76.98	**76.92**	72.15	68.18	67.06

high and suitable for the differentiation between a *benign* and *malware_000* dataset, meaning that we can differentiate most of the goodware and malware using only the characteristics available in the PE32 header. Furthermore, we can see that the two datasets *malware_000* and *malware_207* are similar, and the extracted features do not provide high classification accuracy. Consequently, malware samples could be from the same categories and families. Neural Network was used with 3 hidden layers considering this as a non-linear model. The experiments were performed using 5-fold cross-validation.

2. **Bytes n-gram**

Bytes n-gram is a very popular method for the static analysis of binary executables. This method has one significant benefit: in order to perform analysis, there is no need for previous knowledge about file type and internal structure, since we use it in raw (binary) form. For feature construction, we used randomness profiles that were first presented in the work by Ebringer et al. [133]. They proposed a method of measuring the randomness of certain parts of an executable in order to distinguish between different types of packers used by malware developers. There are two main methods proposed by authors: *fixed sample count*, which generates a fixed number of randomness profiles regardless of file size and *sliding window algorithm*. In this method, each file was represented in a hexadecimal way. The frequencies of each byte were counted, and then a Huffman tree for the whole file was built. Then, using a window of fixed size and moving it on fixed skip size,

the randomness profile of each window is calculated. Randomness profile is a sum of Huffman code lengths of each byte in a window. Thus, the lower the randomness in a particular window, the larger will be the randomness profile.

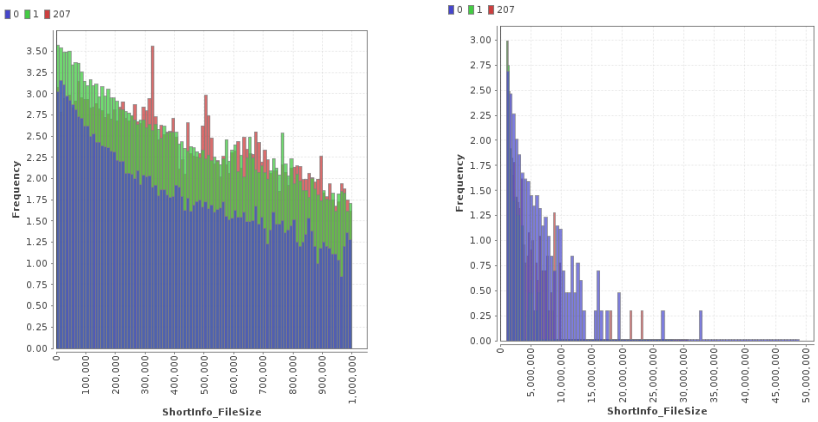
The author's application of this method is not the only one that exists. It is also possible to use this algorithm for extracting features for malware detection, and we tried to use it for our task. As a sliding window size, we chose 32 bytes as the most prominent according to [133, 326, 404]. Due to the big variety of file sizes in our dataset, and especially very small minimal file sizes, we chose 30 of the best features (or *pruning size* in terminology from [133]) which are the areas of biggest randomness (the most unique parts) in their original order. These features were fed into the machine learning algorithms mentioned in the Table 4.4.

Table 4.4: Classification accuracy based on features from bytes n-gram randomness profiles, in %

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
			All features				
Bn vs Ml_000	69.9	60.4	76.9	75.6	78.3	78.3	74.8
Bn vs Ml_207	70.3	68.2	75.8	75.6	72.1	71.6	68.2
Ml_000 vs Ml_207	50.1	64.0	68.1	64.7	58.1	60.1	58.2

In the Table 4.4, the results of machine learning algorithms evaluation is present. As we can see, their accuracy with these types of features are not as high as with others. There are several reasons for this fact. The first reason is that the original method was developed in order to work on the packer classification task. This means that it is probably better at finding unique features of packer (that are meta-features by its nature) that could be spread among a large number of files rather than finding similarities between files on the byte level. The second reason for low accuracy is that the method is developed to preserve local details of a file according to Ebringer et al. [133], and the size of file affects locality a lot. In our case, the file sizes vary from around 0.5Kb to 53.7Mb and in the original paper from 3Kb to 191Kb, which is a significantly smaller range as shown in the Figure 4.1. Despite worse results overall, the tendency kept the same: it was easier to distinguish between benign executables and malware than between malware from different time slices. Also, we can see that ANN is better in the *Benign vs Malware_000* dataset, and C4.5 in the *Benign vs Malware_207* and *Malware_000 vs Malware_207* datasets.

Also it should be noted that we did not use feature selection methods as



(a) Less than 1M **(b)** Bigger than 1M

Figure 4.1: Distribution of file size values in Bytes for three classes

in the case with PE32 header features. Both Information Gain and Cfs are not efficient due to the similarity of features and less-than-equivalence in importance for classification process. For the first dataset of files, Information Gain is in the range 0.0473-0.0672; for the second dataset it is 0.0672-0.1304, and for the last is 0.0499-0.0725. Moreover, Cfs produces the best feature subset—nearly equal to the full set. Therefore, we decided to use all features, since there is no subset that could possibly be better than original.

3. **Opcode n-gram**

Opcode n-gram is the combination of assembly instructions from which an executable file is constructed. They were described earlier in Subsection 2. This method has one limitation: in order to gain opcodes, one needs to perform disassembly, which sometimes fails to get the correct opcodes due to different anti-disassembly and packing techniques used in executables (even though we tried to filter out these kinds of files on the dataset preparation stage). Therefore, we extracted the 100 most common 3- and 4-grams from each of the three file classes. Then, we extracted from them a set of 200 most common n-grams: lets call these feature n-grams. For each file within a dataset, we built a presence vector of length 200 where value 1 was assigned if certain n-grams from feature n-grams are present in the top 100 most used n-grams in this file. The Table 4.5 represents the results of feature selection performed on the dataset with 3-grams. We can see that the first two pairs of datasets have a lot of common n-grams, while the selected n-grams for the third pair of datasets are totally different. For Information Gain, a threshold

equal to 0.1 was used for both benign vs malware dataset, while for the last set we used InfoGain equal to 0.02. It can be highlighted that the first two pairs of datasets use the same 3-grams to differentiate between malware and benign binaries, indicating significant differences in the properties of files.

Table 4.5: Feature selection on 3-gram opcode features. Bold font denotes features that are present in both datasets that include benign samples

Benign vs Malware_000		**Benign vs Malware_207**		**Malware_000 vs Malware_207**	
Information Gain					
merit	attribute	merit	attribute	merit	attribute
0.302483	**int3movpush**	0.298812	**int3movpush**	0.042229	pushlcallpushl
0.283229	**int3int3mov**	0.279371	**int3int3mov**	0.039779	movtestjne
0.266485	**popretint3**	0.227489	**popretint3**	0.037087	callpushlcall
0.236949	**retint3int3**	0.202162	**retint3int3**	0.031045	pushpushlcall
0.191866	**jmpint3int3**	0.193938	**jmpint3int3**		
0.134709	callmovtest	0.108580	retpushmov		
0.133258	movtestje				
0.115976	callmovpop				
0.114482	testjemov				
0.101328	poppopret				
0.100371	movtestjne				
Cfs					
attribute		attribute		attribute	
movtestje		movmovadd		pushpushlcall	
callmovtest		retpushmov		movtestjne	
callmovpop		xormovmov		movmovjmp	
retint3int3		**callmovtest**		jecmpje	
popretint3		**popretint3**		cmpjepush	
pushmovadd		**pushmovadd**		pushleacall	
int3int3mov		**int3int3mov**		callpopret	
callmovjmp		**callmovjmp**		leaveretpush	
jmpint3int3		**jmpint3int3**		pushmovadd	
int3movpush		**int3movpush**		pushcalllea	
				callpushlcall	
				callmovlea	
				pushlcallpushl	
				movmovmovl	
				calljmpmov	

This data was passed to machine learning algorithms; results can be seen on Table 4.6 and Table 4.8. We can see that among all the performed ML methods, C4.5 performs well and has the highest accuracy in almost all experiments. Also, feature selection significantly reduced the number of n-grams from 200 down to 10-15, while overall accuracy on all methods did not drop significantly. In fact, Naive Bayes performed even better than can be explained by reduced complexity of the probabilistic model. Also, NF

showed much better accuracy in comparison to other methods when using all features to distinguish between both malware datasets. This can be explained by the non-linear correlation in the data that are circumscribed in the Gaussian fuzzy patches.

Table 4.6: Classification accuracy based on features from opcode 3-gram, in %

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
All features							
Bn vs Ml_000	83.51	83.52	**95.53**	93.82	94.43	94.51	95.28
Bn vs Ml_207	84.52	84.52	**93.93**	91.84	92.32	92.44	93.20
Mn_000 vs Ml_207	63.73	63.73	81.21	78.64	75.42	76.64	**83.13**
Information Gain							
Bn vs Ml_000	86.74	86.94	90.41	**90.45**	89.98	90.26	84.45
Bn vs Ml_207	86.22	86.22	86.22	86.22	87.46	**87.48**	83.36
Mn_000 vs Ml_207	63.19	62.55	71.19	**71.89**	69.54	67.36	69.14
Cfs							
Bn vs Ml_000	87.79	88.66	91.15	**91.22**	90.90	90.82	85.31
Bn vs Ml_207	86.24	86.33	**89.92**	89.73	89.17	89.34	81.58
Mn_000 vs Ml_207	86.24	86.33	**89.92**	89.73	89.17	89.34	69.25

Furthermore, we investigated the possibility of any correlation between n-grams in files that belong to both benign and malicious classes. We extracted the relative frequency of each n-gram according to the formula $h_{n-gram} = \frac{N^{Class}_{files \in n-gram}}{N^{Class}_{files}}$, where $N^{Class}_{files \in n-gram}$ indicates the number of files in a class that has an n-gram and N^{Class}_{files} is the total number of files in this class. The results for 3-gram are depicted in the Figure 4.2. As a reference, we took the top 20 most frequent n-grams from benign classes and found the frequency of the corresponding n-grams from both malicious classes. The frequency does not differ fundamentally, yet n-grams for both malicious classes tend to have very close numbers in comparison to benign. Moreover, there is a clear dependency between both malicious classes. We also can notice that most of the features selected from the two datasets that include benign samples are same. This highlights the reliability of the selected 4-grams and generalization of the method for malware detection.

Furthermore, we also investigated 4-gram method on the files and extracted 200 features. Later, feature selection was performed with the results presented in the Table 4.7. Similar to the 3-grams features selected in the Table 4.5, one can see that the first two pairs of datasets have a lot of common features, while the last one provides a significantly different set. As in the case with 3-grams, we used Information Gain with a threshold equal to 0.1 for both

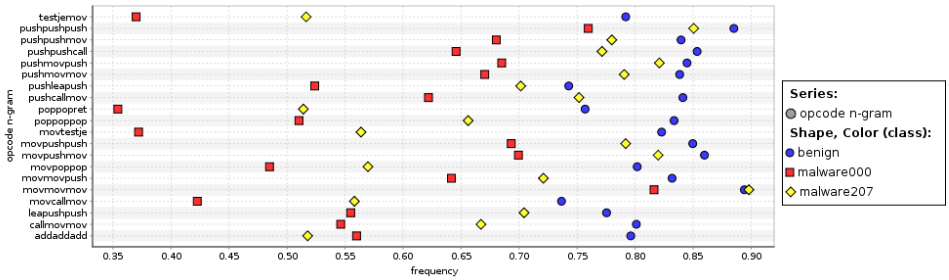


Figure 4.2: Distribution of the frequencies of the top 20 opcode 3-grams from the benign set in comparison to both malicious datasets

benign vs malware datasets, while for the last set we used InfoGain equal to 0.02, which seemed reasonable with respect to number of selected‘ features.

The classification performance is given in the Figure 4.8. As we can see from the results, 3-grams can show a bit better result than 4-grams in case of distinguishing between Benign and Malware000 or Benign or Malware207 with a C4.5 classifier. At the same time, 4-grams are better in order to distinguish between two malware datasets, again with a C4.5 classifier. We can conclude that the results are quite good, and can be used for malware detection. In our opinion, results can be improved in the case of extracting more features and the usage of relative frequencies rather than a pure vector that indicates presence.

Contrary to 3-grams, we can see that the histograms of 4-grams have fundamental differences when it comes to malicious and benign sets, as is depicted in the Figure 4.3. We can see that the frequencies that correspond to malware_000 and malware_207 datasets are nearly similar and are far from the frequencies detected for benign class. Moreover, there is a clear and strong correlation between the two malware datasets. So, we can state that in case of probabilistic-based models like Bayes Network and Naive Bayes, the classification could be a bit better due to differences in likelihood of appearance, which can be also found in the Tables 4.6 and 4.8.

4. **API call n-grams**

   API calls n-grams is the combinations of specific operations invoked by the process in order to use functionality of operational system of the host. Statically, they could be extracted with different debug tools. For our study, we used *peframe*, which can extract API calls from PE32 files. As was known from previous studies described in Section 2 the bigger n-gram size is the

Table 4.7: Feature selection on 4-gram opcode features. Bold font denotes features that present in both datasets that include benign samples

Benign vs Malware_000		**Benign vs Malware_207**		**Malware_000 vs Malware_207**	
Information Gain					
merit	attribute	merit	attribute	merit	attribute
0.303209	**int3int3movpush**	0.295427	**int3int3movpush**	0.047452	pushlcallpushlcall
0.295280	**int3movpushmov**	0.286378	**int3movpushmov**	0.045860	movpoppopret
0.285608	**int3int3int3mov**	0.266966	**int3int3int3mov**	0.044750	jepushcallpop
0.258733	**popretint3int3**	0.229431	**jmpint3int3int3**	0.044573	callpushlcallpushl
0.241215	**poppopretint3**	0.224318	**poppopretint3**	0.038822	cmpjepushcall
0.233205	**jmpint3int3int3**	0.210289	**popretint3int3**	0.035731	pushcallpopret
0.220679	**retint3int3int3**	0.170367	**retint3int3int3**	0.030460	pushcallpopmov
0.185178	**movpopretint3**	0.148442	**movpopretint3**	0.028564	movcmpjepush
0.151337	**movpushmovsub**	0.116760	**movpushmovsub**	0.025813	cmpjecmpje
0.125703	pushcallmovtest	0.103841	**movpushmovpush**	0.024372	leaveretpushmov
0.104993	**movpushmovpush**	0.102730	**movpushmovmov**	0.023374	pushpushpushlcall
0.104416	**movpushmovmov**			0.022312	pushcallpoppop
				0.021929	movtestjepush
				0.020003	pushpushleapush
Cfs					
attribute		attribute		attribute	
incaddincadd		addaddaddadd		leaveretpushmov	
movpushmovsub		movmovpushpush		callmovtestje	
jmpmovmovmov		**movpushmovsub**		jepushcallpop	
pushcallmovtest		**pushcallmovtest**		pushlcallpushlcall	
int3int3int3mov		**int3int3int3mov**		pushpushpushlea	
movpoppopret		movxormovmov		jecmpjecmp	
jmpint3int3int3		pushlcallpushlcall		movpoppopret	
movpopretint3		**jmpint3int3int3**		pushcallmovpush	
int3int3movpush		movpopretint3		pushmovmovcall	
int3movpushmov		**int3int3movpush**		movpopretint3	
poppopretint3		**int3movpushmov**		cmpjepushcall	
addpushpushpush		**poppopretint3**		movleamovmov	
pushpushcalllea				movmovjmpmov	
				pushpushcalllea	
				retnopnopnop	
				movaddpushpush	
				subpushpushpush	

lowest accuracy it is possible to gain. The reason for this is that single API calls and their n-grams are present in files in fewer amounts comparing to, for example, opcode n-grams. Sometimes, a single executable can have less than several dozens of API calls. During the extraction process, *peframe* fails to extract API calls on some files due to internal errors or file antidebug features. After extracting API calls, we combined them into 1- and 2-grams. For each task, we selected the 100 most frequent features in particular class and combined them into 200-features vector. In the Tables 4.9 and 4.10

Table 4.8: Classification accuracy based on features from opcode 4-gram, in %

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
All features							
Bn vs Ml_000	86.92	86.92	95.31	93.73	94.28	94.23	**95.54**
Bn vs Ml_207	86.84	86.84	93.33	91.71	92.03	92.04	**93.75**
Ml_000 vs Ml_207	64.90	64.90	**81.58**	78.98	74.98	75.77	78.80
Information Gain							
Bn vs Ml_000	87.79	87.89	**91.48**	91.45	91.31	90.84	85.74
Bn vs Ml_207	84.64	84.57	**87.84**	87.83	87.25	87.70	48.67
Mn_000 vs Ml_207	62.73	63.20	69.96	**70.25**	68.40	67.24	68.90
Cfs							
Bn vs Ml_000	89.63	89.63	91.51	**91.52**	91.52	90.76	84.95
Bn vs Ml_207	86.41	86.64	89.36	**89.48**	89.16	89.12	81.13
Mn_000 vs Ml_207	66.28	66.17	72.00	**72.27**	68.96	69.17	69.32

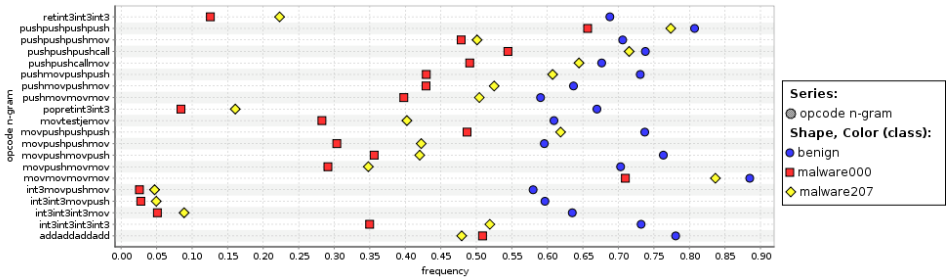


Figure 4.3: Distribution of the frequencies of top 20 opcode 4-grams from benign set in comparison to both malicious datasets

results for machine learning evaluation on API call n-grams are present.

Table 4.9: Classification accuracy based on API call 1-gram features, %

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
All features							
Bn vs Ml_000	90.79	90.79	93.39	93.47	**93.51**	93.43	82.44
Bn vs Ml_207	87.18	87.18	90.94	91.03	**91.37**	91.23	81.28
Ml_000 vs Ml_207	66.19	66.2	**78.44**	77.09	73.33	72.77	73.55

As we can see from the tables ANN, kNN and C4.5 are the best classifiers, similar to previous results. It is also more difficult to distinguish between files from *malware_000* and *malware_207*. We gained quite a high accuracy, but it is still lower than in related studies. This could be explained by the size of datasets: in other studies, datasets usually consist of several hun-

Table 4.10: Classification accuracy based on API call 2-gram features, %

Dataset	Naive Bayes	BayesNet	C4.5	k-NN	SVM	ANN	NF
			All features				
Bn vs Ml_000	86.54	86.55	90.88	91.53	**91.96**	91.85	75.24
Bn vs Ml_207	81.94	81.91	87.84	**88.82**	88.31	87.81	83.61
Ml_000 vs Ml_207	62.31	62.31	**73.69**	73.17	70.27	69.45	70.08

dreds or thousands of files while our dataset has more than 110,000 files in it. After analysing feature selection results, we decided not to include them in the results section since most of the features are similar in terms of distinguishing between malware and goodware. This means that there are a large number of unique API calls that can be found once or twice in the file, in contrast to byte or opcode n-grams.

Furthermore, we also studied the difference in frequency distributions among API calls. The Figure 4.4 sketches extracted API 1-grams from three datasets. One can see that there is a significant spread between the number of occurrences in benign classes in contrast to both malicious datasets. On the other hand, the results for both malware datasets are similar, which indicates statistical significance of the extracted features. It is important to highlight the largest scatter in frequencies for the functions *memset()*, *malloc()* and *free()*. It makes more sense for the developers of benign software to carefully allocate and clean memory that the program uses to make it more efficient, while malware producers mostly do not care about this issue since it is not an important factor for the malicious payload. On the other hand, malicious programs tend to use *GetProcAddress()* the function more often for retrieving the address of any function from dynamic-link libraries in the system.

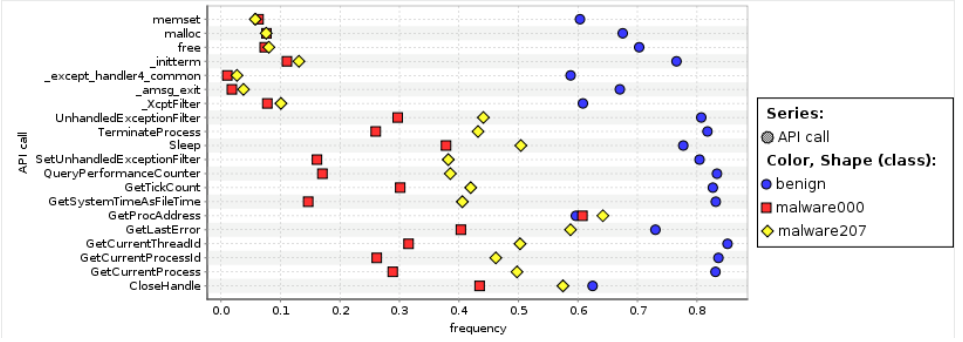


Figure 4.4: frequencies of 20 most frequent API 1-grams for three different datasets

Relevance of Collected Datasets

Along with the study of ML application for malware detection, we also performed a study to check whether the collected datasets actually represent the real distribution of the malware and goodware. One of the credible sources for this information is the so-called "Compile Time" filed that can be extracted from the PE32 header. A previous section described one benign and two malicious datasets that were collected for this study. The Figure 4.5 represents a log-scale histogram of the compilation time for the benign dataset. Since our target is the Windows NT family, we performed an exploration of the history of the Windows OS [279, 440]. Taking into consideration the OS timeline, we found some consistencies. Microsoft has produced a large number of versions of Windows operating systems starting from Windows 1.0 in 1985 and finishing with the recent Windows 10 released in summer 2015. To start with, **Windows 3.1** was originally released on April 6, 1992, and our plot indicates the biggest spike around the first part of the year 1992. Later on in the 90's, **Windows 95** arrived on 24 August 1995, while the next **Windows 98** was announced on 25 June 1998. Further, the 2000's marked the release of **Windows XP** on October 25, 2001. This OS had great success and was installed on PC by home users as well as industrial customers like ATM machines and supermarket systems. The following phases of the plot describe the releases of **Windows Vista** on 30 January 2007 and **Windows 7** on 22 October 2009. Following that, the next popular version appeared on 26 October 2012 and was named **Windows 8**. Finally, the latest major spike at the end of 2014 corresponds to the fact that **Windows 10** was unveiled on 29 July 2015. Spikes over dates of major Windows NT versions, also followed by a significant increase of number of files coming from updates and corresponding software releases right after corresponding version of Windows are released. However, what is more interesting is that we did not get any binary from Windows 1.0 or Windows 2.0, which means that binaries from such OS are not included in latest versions. Finally, there are no binaries from Windows 1.0 (20 November 1985) and Windows 2.0 (9 December 1987), meaning that these systems are 16 bit architecture, while Windows 3.1 and later use 32bit architecture, or PE32 files. Considering this, we think it is reasonable to concentrate on the Windows versions that have been in use over last two decades, such as 2000, Vista, XP, 7, 8, 8.1 and 10.

Further, the compilation time distribution for the first malware dataset *malware_000* is given in the Figure 4.6. We can clearly shows that the release of another Windows version always causes an increase in the cumulative distribution of malware samples in the following 6 to 12 months. The gap between release and spike is justified by the fact that attackers take time to study possible vulnerabilities and create a malicious code. It can be seen that the release of 32bit Windows 3.1

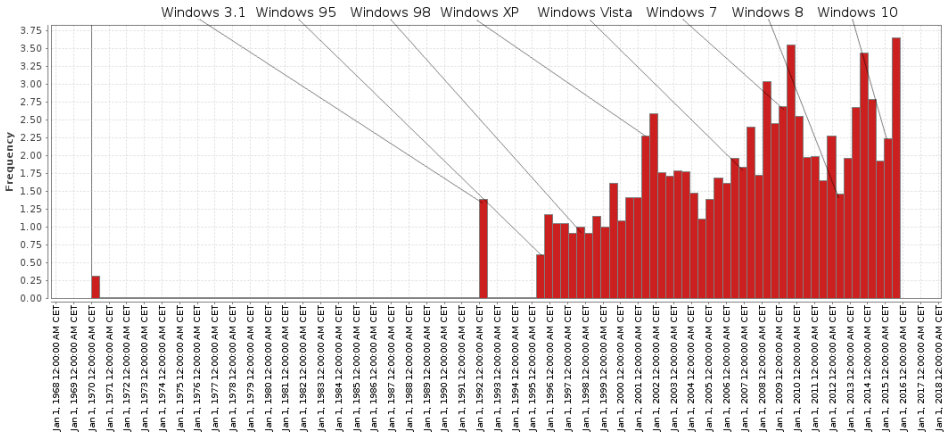


Figure 4.5: Log-scale histogram of compilation times for *benign* dataset

caused a spike in a number of malwares. After this, the number of malware compiled each year is constantly growing. Another increase can be observed in the second half of the 2001 year which is relevant to the release of the most popular Windows XP. Since this dataset was collected early in 2012, this means that the collection of malware samples in most cases covers Windows versions up to Windows 7 released on 2007.

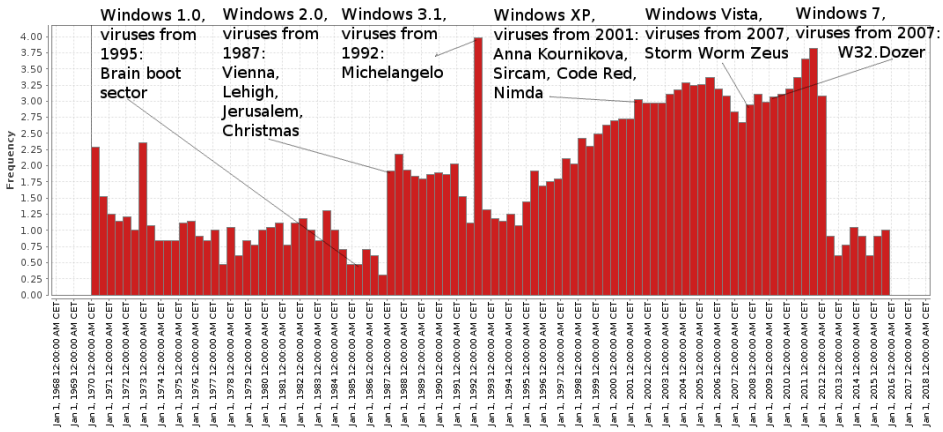


Figure 4.6: Log-scale histogram of compilation times for *malware_000* dataset

Considering the fact that MS DOS was released in 1981, it seems as though compilation times before this day look fake or intentionally obfuscated. On the other hand, the dataset *malware_000* can not have dates later than June 2012. How-

ever, there are quite many files later compiled just before 2016, meaning that the compilation dates in these files are tampered with. Furthermore, the *malware_207* dataset seems to have more recent versions of malware, and the number obviously increases towards the year 2016 staring from the mid-90's. A number of outstanding viruses have emerged since release of Windows OS. It worth mentioning the Brain virus, primarily designed for MS-DOS, which infects the boot sector and is also compatible with IBM PCs [439]. Additionally, Jerusalem hit the world in 1988, being destructive for most of the executable files found on the computer [436]. Later on, Michelangelo was expected to create a massive attack in 1992 by rewriting the first 100 sectors of hard disk. At this point, we clearly see that most of the viruses were written for MS-DOS as a basis of Windows OS, while Windows was considered a graphic addition. One of the most well-known trojans is Storm, which infected about 2 million computers in 2007 through email attachments, according to Moloseciv [282]. In 2007, Zeus botnet also struck the world and was mostly deployed on MS Windows OS intending to steal sensitive information and create botnets [439]. The period before the release of Windows 7 was marked with several waves of July 2009 cyber attacks deploying W32.Dozer that erases information and prevents a computer from being rebooted [439]. In comparison to the first malware collection, there are files also covering Windows versions after Windows 7 as shown in the Figure 4.7. Also, we see that the largest number of malware samples were compiled in the range from 2012 up to as recent as 2016. This is caused by the fact that Microsoft released Windows 8, 8.1, and finally ended up with Windows 10 around that time. Also, these two malware datasets seem to have more reliable malware samples considering the reduced amount of malwares compiled before release of Windows 1.0.

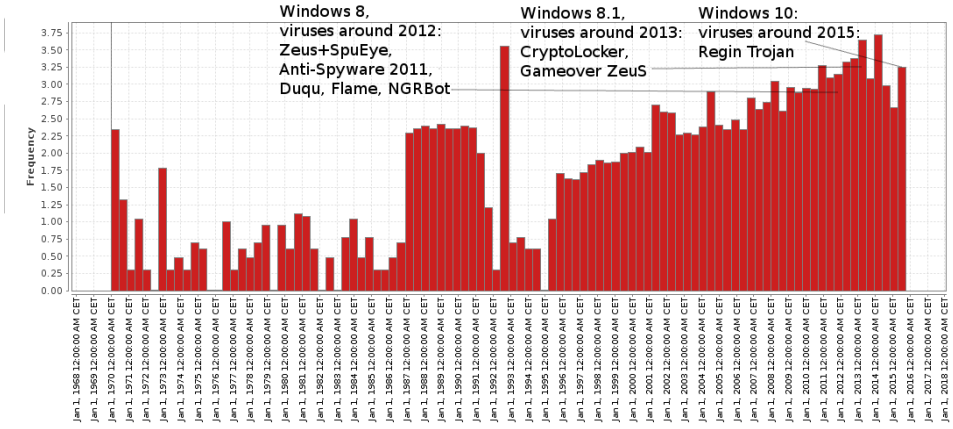


Figure 4.7: Log-scale histogram of compilation times for *malware_207* dataset

Later, following the release of Windows 7 and Windows 8, a number of malware were developed in addition to the already mentioned viruses. Duqu was discovered in 2011, which provided a large functionality to attackers by exploiting a number of vulnerabilities in MS Windows [282, 439]. Flame then appeared in 2012 while being referred to Duqu creators as studied by Milosevic [282]. Finally, Regin Trojan was discovred in 2014 just before the release of Windows 10 and affected computers via spoofed web pages and is supposedly designed for cyber espionage [439]. Thus, it can be concluded that benign samples reflect the history of MS Windows OS, while the distribution of malware tends to cover the development of corresponding viruses. Additionally, malware developers used to set false and sometimes infeasible compilation times to obfuscate and deceive malware analysts.

Summary: Above, we presented a tutorial and a survey on the application of Machine Learning models for Portable Executable malware detection using static analysis. We have considered the different ways of extracting the most relevant attributes from static characteristics of the executable files. Then, we provided an overview of the publicly available dataset, ML methods with corresponding implementations. Additionally, we saw that static-based detection using ML can be used as a fast and reliable tool considering the different characteristic of PE32 that eliminates a need to execute it and may give a tentative understanding of the purpose of a executable or library. Furthermore, we believe that this approach is also applicable in Cyber Threat Intelligence since compromise indicators can be retrieved from static features. Below, we will provide an overview of the features and accuracy of ML methods with respect to feature selection process. Based on the extensive experiments performed, one can see that C4.5 and k-NN in most cases perform better than other methods. SVM and ANN showed good performance on some datasets. On the other hand, Bayes Network and Naive Bayes have poor performances compared to other ML methods. This can be explained by negligibly low probabilities, especially present in a large number of features such as opcode and bytes n-grams.

Meanwhile, there are a growing amount of benign files around 2009 as shown in the Figure 4.5 which reflects Windows Vista - Windows 8 releases. Moreover, the accumulated growth of malware samples is seen around 2008-2009 in the *malware_00000* dataset in the Figure 4.6. Similarly, the more recent dataset *malware_00207* has similar tendencies show in the Figure 4.7. As a result, the increased number of publications per year indicates the importance of the static analysis of Portable Executables using machine learning methods. Thus, we can conclude that most of the features extracted by means of static analysis are suitable for robust malware classification. One main discovery was related to compilation time of

binary files. Goodware seem to follow releases of Windows OS, while malware developers used fake or sometimes infeasible compilation times for obfuscation. We noticed that quite many malware samples presumably had been compiled before release of MS-DOS OS in the 1980's, which should be doubted.

4.2 Windows Portable Executable 32 Bit: A Novel Multinomial Malware Collection[2]

There are two main approaches to analyzing malware samples: static and dynamic analysis. *Static analysis* includes scanning files to collect relevant raw characteristics from the file, while *dynamic analysis* reveals behavior characteristics by executing them in an isolated environment, as studied by Ravi et al. [331] for multiple malware families. As a result of obfuscation, the different methods utilized by malware to avoid detection, some dynamic analysis methods require user interaction to trigger the malicious behaviour. Despite the fact that dynamic analysis could be comprehensive, this would require much more time and resources than static analysis methods. Because of this, we consider static analysis to be more appropriate for our project, taking in mind the large number of samples and time constraint. The main goal of this section is to study how static analysis of PE32 and classification through SC methods can facilitate large-scale detection of malware families and malware types. A novel dataset has been composed during the experimental phase. It will be used further in different contexts with respect to the aforementioned challenges in the Chapter 2.

Most antivirus scanners use signature-based and heuristic-based detection methods, where they search for known patterns in executable code as well as the hash sum of the file against known malicious files in a database. A limitation of signature-based methods are that the malware must be obtained and analyzed before the antivirus vendors can update their databases, as described by Ye et al. [449] and Kolter et al. [231]. Far more flexibility is provided by ML-based methods, such as nature-inspired SC methods that allow the building of inexact models from incomplete and complex data. Das et al. [111] gave an overview of how SC methods can be used in different areas, including in the area of Information Security.

4.2.1 Dataset

There was a malware collection initiative that took place within the NTNU Digital Forenisc research group[3] during Summer 2015 using the following sources: *maltrieve* [269] that extracts recent modern malware samples, the 10 first archives available at *VirusShare* [17] starting from *VirusShare_00000.zip*, the collection

[2]The main ideas of this setion are published under the contributions [167, 379, 381]

[3]https://testimon.ccis.no

from *VxHeaven*, and files that students share within the group. After thorough analysis and filtering, we derived PE32 files and removed other types of files. Initially, we ended up with 407,741 malware samples. This will be further named as a **novel PE32 malware collection**. It can be considered large-scale suitable enough for our experiments, since datasets used in the studies before are mostly small sets with thousands and tens of thousands of samples [333, 458, 385, 82].

Since we targeted a static analysis, it was decided to extract as much of the raw characteristic as possible to facilitate large-scale malware analysis. The two main sources that we used were *PEframe* [57] and *VirusTotal* [18]. *PEframe* presents a comprehensive set of attributes that can be found in the PE32 header. There has been some work that demonstrated the possibility of identifying malware using such information. *VirusTotal* presents scan results from over 65 anti-virus databases, information about possible packers and compressors in addition to basis PE32 headers data. All data from the *VirusTotal* were gathered using Private API. Moreover, we used standard Linux tools to retrieve various file characteristics, e.g. the size of different sections, strings, and entropy. Finally, we created a MySQL database with raw characteristics of all PE32 executables filtered out from a heap of gathered earlier samples. The overall process can be described as following:

Malware samples collection

The resulting malware collection consists of a number of samples from students, the first 10 archives of *VirusShare* [17], and files from *VxHeaven* [20] were collected in addition. *VirusShare* offers number of archives, out of which we used *VirusShare_00000.zip–VirusShare_00009.zip* that covers a range of collected samples from *2012-06-15* to *2012-09-15*, and in a size range from 13GB up to 85GB. Each of these archives includes 131,072 files equally. Also, there was a collection done by *Maltrieve* [269] with thousands of recent malware samples available in the Internet. To pre-process files, we renamed them with *md5* values and filtered out all files other than PE32 by Linux ‘file‘ command. Overall, we reduced the number of files by eliminating duplicates and non-PE32 files. Thus, we ended up with 407,731 malware, yet some of them will be eliminated. The total size of all the executables is 136GB out of all archives from *VirusShare*, which were 378GB.

Characteristics acquisition

NF is designed to handle numerical features as inputs. Therefore, static analysis is applied on the collected dataset to extract the corresponding characteristics for future feature construction. We targeted a static analysis and not a dynamic one due to the large number of collected samples and specific requirements for malware execution. The two primary objectives: (i) get a set of static characteristics for

each file, including those available in the PE32 header, (ii) retrieve the most likely malware category and family name for each file.

PEframe presents a comprehensive set of attributes that can be found in the PE header of a format showed in the Listing 4.1:

Listing 4.1: Example of PEframe output

```
"Anti Debug": [
        "GetLastError",
        "TerminateProcess",
        "UnhandledExceptionFilter"
    ]
"Suspicious API": [
        "GetCurrentProcess",
        "GetCurrentProcessId",
        "GetTickCount",
        "Sleep",
        "TerminateProcess",
        "UnhandledExceptionFilter"
    ]
```

There were some works before showing that it can be possible to detect malware using such headers information.

VIRUSTOTAL reports are scan results from over 60 anti-virus databases, information about possible packers, *exiftool* data, and detected packers versions in addition to basic PE header data. An example of the output is given in the Listing 4.2. In order to retrieve information from the database, one has to use API that sends a *md5* sum and retrieves JSON-formatted report. The daily quota was 5,760 requests, so we asked for academic access to be able to process 400k samples in reasonable time. It took more than a week to get all the data collected. Despite the fact that it contains information from 60 anti-viruses, it still can be considered as subjective analytical results that may indicate different malware families for the same malware.

Listing 4.2: Example of VirusTotal report

```
[Microsoft] => stdClass Object
(
        [detected] => 1
        [version] => 1.10401
        [result] => PWS:Win32/OnLineGames
        [update] => 20140404
)
```

Finally, we used standard Linux tools to retrieve more file characteristics, e.g. size of different sections, strings, and also entropy. We created a MySQL database that contains the raw characteristics of the PE32 windows executables (all malware) filtered out of the mess of different malwares that were present in gathered earlier sets. Fields in the initial SQl table are as follows:

1. *md5* - unique id also a key in *VirusTotal*.
2. *virustotal_file_report* - VirusTotal report [18].
3. *virustotal_file_behaviour* - Cuckoo data API [18].
4. *virustotal_file_network_traffic* - network communication.
5. *peframe* - *PEframe* report, JSON-formatted.
6. *file* - output of the Linux command 'file'.
7. *strings* - command 'strings' (ASCII).
8. *size* - Linux command 'size' of the format:

```
text   data  bss   dec     hex    filename
10518  2044  3424  110650  1b03a  /bin/ls
```

9. *file_entropy* - entropy of the file, may indicate the degree of encryption. A typical range is from 0.0 up to 8.0.
10. *size_of_file* - size of the file in bytes.

It took about a week to collect information from *VirusTotal* database and *PEframe* output. The behavioural analysis as well as network interaction was not available from *VirusTotal* for most of the files at the time of the dataset composition, so we excluded it from the feature extraction stage. We also contributed to the VirusTotal *database* since there were around 200 malware samples not present there.

Feature Extraction

Before meaningful automated analysis can be executed, we must extract numerical features by performing a preliminary manual processing of raw static characteristics. On this step, we (i) transform the gathered raw characteristics into *numerical features*. To accomplish this task, we processed *VirusTotal* and *PEframe* reports. The content from both sources was the form of a serialized JSON object. Therefore, we focused on extracting as many relevant numerical features as possible from this structure. Extracted features are given in the Table 4.11.

Afterward, (ii) we extracted the corresponding *malware types and families* names from *VirusTotal* output. Furthermore, feature selection was performed to determine which features contribute most to classification. Feature extraction is the next important step that we pursue while working on malware samples collection. Classifying families and categories were the main objectives and challenges when we created the dataset. Our initial hypothesis was to utilize and parse scan reports from *VirusTotal*. A quarter-century ago, an organization named CARO (Computer Antivirus Research Organization [4]) was founded to establish common naming schemes to be used by anti-virus vendors. One may notice that there exist a high number of categories (like *trojan, backdoor*, etc) and families (like *Poison, Ramdo*, etc) that are commonly defined by the Information Security community. Our first thoughts were to apply a majority voting scheme for families in case of conflicting names, but all vendors used an inconsistent and variable naming approach that make it infeasible to classify data. Moreover, hardly more than Microsoft followed the standard, so we considered only files where Microsoft indicated the malware properties, which reduced the number of files from 400k to 328k. For example, if the malware scan result by anti-virus is ADWARE.WIN32.CONDUIT.M, then the category is ADWARE and malware family is CONDUIT, while the platform architecture is WIN32. The most comprehensive collection of malware family names for OS Windows was done by Microsoft [278], which assembled 246 different items. From malware collection, we managed to extract 10,362 families and 35 categories. Finally, a Python script was written to first extract the data, and then to put it into a new database table.

In general, one can say that the challenge with categorization is that malware categories had commonly accepted names [123], while families were partially invented by malware analysis, which first found a specific species in the wild and analysed it [217]. After filtering out malware that does not contain structured categorization information from VirtusTotal, there have been found 10,362 malware families and 35 types in 328,337 collected samples according to MS antivirus reports. The details will be discussed in the following section. The top 35 labels for both classes are given in the Table 4.12. It can be seen that the large majority of malware samples lies only in a few malware categories, while other categories include only a few malware samples.

4.2.2 Static Analysis in Hard & Soft Computing Models

As was mentioned earlier in 2 there are considered two main sets of methods in Machine Learning, Hard Computing and Soft Computing. HC includes the conventional methods dealing with crisp logic, while Soft Computing is more tolerant toward data errors and incompleteness than are present in real-world data. The results of multinomial classification are given below.

Experimental Design

As was mentioned earlier, we gathered a novel large-scale malware dataset. Initially, (i) we reduced the number of files by eliminating duplicates and non-PE32 files. After a thorough analysis and filtering, we derived all possible types of PE32 files designed for MS Windows and removed other types of files. Overall, we ended up with 407,741 unique malware samples that are PE32 files, as was mentioned earlier in this Section. The total size of all the executables is 136GB despite the fact that all archives from *VirusShare* occupied 378GB. Furthermore, (ii) we performed a raw characteristic acquisition from *PEframe* and *VirusTotal* and obtained a MySQL table of 19.5GB that later resulted in 98.7MB of numerical features totaling 328,337 malware samples. The number is lower since we only used samples that contained a CARO-formatted report from Microsoft anti-virus. Consequently, after filtering and pre-processing we ended up with 328k samples that consisted of 10,362 malware families and 35 types. Our preliminary study showed that such a high number of families makes the application of most ML methods infeasible. Therefore, the decision was made to create the following subsets limiting the most frequent types and families to decrease the number of highly-imbalanced classes.

1. *10 families* containing 74,655 samples, 11.8 MB.
2. *100 families* containing 227,191 samples, 36.3 MB.
3. *500 families* containing 292,596 samples, 46.7 MB.
4. *10 types* containing 310,355 samples, 50.5 MB.
5. *35 types (full set)* containing 328,337 samples, 53.5 MB.

The Figure 4.8 presents the distribution of the smallest sets.

Choice of Soft Computing methods

As mentioned earlier, SC is a set of nature-inspired methods that are designed to handle complex data and produce inexact solutions that can be also interpretable. Singh et al. [394] sketched how the variety of SC methods can be applied for malware detection. In this thesis, we concentrate on the following SC methods in our experiments.

- *Naive Bayes* is a simple classifier built on the assumption that features are independent from classes as described by Rish [335]. This method performs well on nominal features compared to other classifiers.

- *Bayesian Network* is based on the conditional probabilities of features in a directed acyclic graph as presented by Friedman et al. [154]. Probabilities of the observable variables are then computed.
- *MultiLayer Perceptron* is a type of Artificial Neural Network that simulates how the human brain neurons learn from observations according to Kononenko et al. [232]. This is achieved by using multiple hidden layers for better representation of non-linear data.
- *Support Vector Machine* learns from data by determining the optimal hyperplane to best separate the data. The better accuracy is achieved by using kernels in SVM that can project the data to a higher dimension feature space to create such a hyperplane according to Hearst et al. [181].

Feature Selection methods

Proper feature selection will contribute to a higher classification accuracy and reduction of computational complexity, which in turn will increase the overall classification performance as suggested by Kononenko et al. [232]. After the examination of the current state of the art in feature selection methods, we decided to use the following methods:

- *Cfs* selects features with a high correlation to classes and disregards features with a low correlation. Experiments by Hall [175] show that it can easily eliminate irrelevant, redundant, and noisy features.
- *Information Gain* ranks features by entropy with respect to each class as presented by Roobaert et al. [338].
- *ReliefF* ranks features from how well they help to separate classes of data samples that are close to each other. This is an extension from the two-class Relief algorithm. The algorithm was designed to perform on data with missing values as given by Kononenko [232].

Our main motivation is also to see how the feature's merit influences classification and whether we can judge about the utility of the feature in malware analysis as a compromise indicator.

Performance Evaluation

The following metric was used to evaluate the performance of both methods using overall accuracy $Acc = \frac{N_P}{N_S}$, where N_S - number of data samples and N_P - number of properly classified samples according to $max - min$ principle [233] in Mamdani-type rules.

Results & Analysis

During this research, we performed more than a hundred different experiments. Since this is a novel work, we figured out that many of the methods could not be successfully performed due to the large model size in computer memory. Also, we did not consider the results of experiments that took more than a week to execute. Below, achieved accuracy is given for SC and HC methods on different sets of malware and features.

Feature Merits

All experiments on feature selection and classification were performed using cross-validation to see how well the methods can handle unlabeled data samples.

Feature selection resulted in a set of various features. The results from *Cfs* contain only the selection features and not the feature's merits. This is because *Cfs* utilizes another search algorithm than *Information Gain* and *ReliefF*. Also, we can say that the *ReliefF* method shows considerably smaller average merit for each feature than was achieved by *Information Gain*. This suggests that we have to use more features with much smaller thresholds to be able to achieve good results. The numbers of extracted features per method are shown in the Table 4.13. For *ReliefF*, we used only features with merit $\geq 10^{-3}$. For *Information Gain* we used features with merit ≥ 0.1 for types, and ≥ 0.5 for families respectively. *Cfs* is not a ranking-based method and only generated the final subset of features.

Furthermore, we can state that *ReliefF* in general gives a larger number of features from a less-significant range of the feature's merits. For example, on a set of the 500 most frequent families the merit range for *ReliefF* is 0.0-0.063 against 0.0-2.188 in *Information Gain*. The Table 4.14 shows the features selected by both *Cfs* and *Information Gain* on each data subset. The features in bold were selected by both methods for three subsets of families and two subsets of types respectively. An interesting observation considering the features selected for family classification is that the features selected converge with the increase in the number of classes.

Two important features that we can highlight are *"pe_api"* and *"vt_sections"*. Both features were selected by all three methods for feature selection. What we can derive from this is that the different malware families differ in the numbers of API-calls made in the file and in the number of PE32-sections in the files. Previously, we discussed the structure of a PE32 file. There are a number of sections mandatory for a PE32 file to run, while the total possible number of files is much greater (maximum 2^{16}-1 = 65,535), as studied by Kath [218]. The number of API calls within the different families has a certain variety as well, from which we can

tell that similar malware families will have a similar number of API calls. Since the different families have different capabilities and functionality, we conclude that our assumption that API calls will be of interest to labelling malware is supported after this observation.

Classification performance

Performance was estimated using a 5-fold cross validation. Also, experiments with the resampling filtering method showed an improvement in accuracy, though we did not include these in the final results. Along with the SC method, we decided to compare accuracy of a set of HC methods such as symbolic reasoning, as was also studied by Abraham [40]. The comparison of the achieved accuracies are shown in the Table 4.15. The highest accuracy for each subset is highlighted with respect to feature selection methods. To give more wide coverage, we only considered *Accuracy* as performance metric for all methods. *Bayes Network* shows considerably good performance among SC methods. *MLP* performs much better on smaller sets, which might indicate a need to use a higher non-linearity, for example Deep Neural Network. Furthermore, *Naive Bayes* gives a huge error rate on all sets, meaning that the method only works well on nominal-valued features. On the other hand, we can see that in general, HC methods such as tree-based *C4.5* and *Random Forest* tend to have better performance on defined problems of malware classification. However, for the 500 families dataset, *C4.5* can result in 32,904 leaves with a total size of the tree equal to 65,807. For 35 malware types, this method produces 33,952 leaves with a total size of the tree equal to 67,903. In real life however, this model is not practical.

Note that we were able to train *SVM* only on the smallest dataset (10 families). Also *Random Forest* exhausted available memory on the largest dataset (500 families).

Summary: In this paper, we investigated a large-scale malware detection using Soft Computing methods based on the static features extracted from PE32. It is important not only to distinguish between benign and malware files, yet also to understand what kind of malicious file it is: malware family and type. To explore this problem, we created a novel datasets of malware features with respect to families and types using publicly available sources and tools. At this point we can see that among SC methods, Bayes Network performs much better than others, while Naive Bayes is the worse one. HC can produce reasonable results and better accuracy, yet the resulting model is incredibly large for C4.5 and Random Forest. The last also failed to train on the largest dataset. Feature selection mostly cannot provide a consistent improvement. Moreover, SVM is not scalable at all. This extensive study contributes to the area of Soft Computing and Information Security

also by giving an insight for malware analysts on how to detect malware types and families extracted from static analysis.

4.2.3 Improved Multi-Class Neuro-Fuzzy for Static Analysis

Traditional malware classification research considers only binary problems, where a set of benign software samples with a corresponding heap of malware binaries are used to extract features and classify a model respectively. Due to the large number of malware targeting MS Windows however, it is important to understand the differences between various malware categories and families. These have quite different functionalities, infection methods, and impact; among others, these are trojan, worms, keyloggers, etc. In some cases, such malware are used as an initial compromise instrument in the Advanced Persistent Threats (APT) attacks. Even though it might be challenging to detect ongoing APT, one can still identify a used virus sample without execution. Moreover, we believe that the application of fuzzy rules can simplify binary analysis in security labs by narrowing down likely malware category or family of unknown binary file. Our motivation is therefore to perform such a study on the malware samples that are available for Windows OS. Portable Executables (PE) is the file format used in this OS for executables, linked libraries, etc. It contains a compiled program with a PE32 header describing different properties such a lookup tables, addresses and the size of different file sections. According to the official Microsoft manual [427], it covers a large number of architectures, characteristics, and data directories that make it suitable for malware detection.

Taking into consideration the last decade in the history of Windows malware, we can see that this has been the decade where most of the modern malware categories were discovered, in addition to being a decade of almost exponential growth of the number of detected malware. Windows NT OS family (like XP, Vista, 7, etc.) has been widely deployed by both private and public sectors, and in both critical and entertainment infrastructures. Despite some security fixes, the Windows NT family was under attacks due to outdated installed software like Internet Explorer 6 and unpatched versions of the OS. One of the main reasons for such a tendency is also the lack of awareness by the user about security problems in the OS as studied by Symantec Lab.

Another concern regarding malware classification is within the characteristics acquisition. There exist static- and behaviour-based analyses that address the various aspects of PE32 files as we can learn from the SANS report [123]. Despite the comprehensiveness of the dynamic approach, we believe that it can hardly be considered applicable against large-scale collections due to the limitations on OS version and installed software. Static analysis, on the other hand, is fast and does

not require execution. If we look at the previous studies that involve PE32 file formats for MS Windows, we can see that the majority of them target only binary differentiation such as "malicious" and "benign" samples. For example, Markel et al. [264] explored an application of static PE32 header data as primary indicators in the detection of malware and benign files. Hahn [173] extensively studied PE32 header and its efficiency in the detection 103,275 malware samples from a single *VirusShare* archive against 49,814 goodware collected from MS Windows OS. However, there are many malware categories and families that have various characteristics and functionality. Therefore, our idea is to study how the static analysis of PE32 files can facilitate large-scale malware detection into families and categories with the help of the Neuro-Fuzzy method.

Experimental Setup

The main goal of our experiments was to explore the behaviour of the NF method on large-scale multinomial malware families and categories detection problems. This is a novel and quite comprehensive preliminary study and therefore we consider only the 10 most frequent classes and 10 frequent families to have balanced distribution. From the results of this study, we can say that many methods could not be trained in a reasonable time while the number of classes reaches hundreds. Additionally, we decided to compare ANNs with less than 3 layers, so DNN are not within our scope. However, it is important to see whether with higher non-linearity (more layers), it can outperform NF.

Properties of Dataset

We had to overcome significant limitations in standard tools to be able to process such a large malware samples collection, as well as the characteristics dataset. This represents a significant interest to the number of samples and collected characteristics. First, we filtered out entries where tools outputs were not retrieved or were empty. Second, the data pre-processing and exploratory analysis were done using the open-source software *Weka* [149]. To eliminate highly imbalanced classes, we decided to concentrate on the 10 most frequent categories and 10 most frequent families, which are listed below:

- families: *vb, hupigon, vundo, obfuscator, agent, renos, small, onlinegames, vbinject, zlob* - 74,655 samples.
- categories: *trojan, pws, trojandownloader, worm, virtool, backdoor, virus, rogue, trojandropper, trojanspy* - 310,355 samples

To compare with, Ou et al. [303] used only 110,530 protein data samples, 15,000

English letters samples and 60,000 handwritten digits. Thus, both our generated subsets are large-scale and describe two different problems in malware detection.

By visualizing extracted features we can see that some classes have distinguishable patterns and can therefore be described by fuzzy rules. The Figure 4.9 shows an obvious correlation between a number of suspicious API calls (according to *PEframe*) and the entropy of the file (meaning possible encryption of the file content). Moreover, groups in different regions are clearly visible. On the other hand, it is difficult to see specific groups in the dataset when considering the number of suspicious API calls and file sizes in the malware categories dataset as shown in the Figure 4.9. This means that it might be hard to perform such differentiation using only extracted characteristics.

Another important consideration regarding the dataset is whether it actually reflects the modern global distribution of malware samples for MS Windows. By having 400k samples, it is hard to analyse this aspect manually. However, we consider a compilation time from PE32 header as a good relevant indicator. We used *RapidMiner* [14] to depict the distribution of the compilation dates using a year, month, and day stamp as given in the Figure 4.10.

Next, we studied the history of Windows OS [279] and found some consistencies. For example, Windows 3.1 was originally released on April 6, 1992 and our plot indicates the biggest spike around the first part of the year 1992. After this, the number of malware compiled each year is constantly growing. Then, another increase can be observed in second half of the year 2001, which is relevant to the release of the most popular Windows XP on October 25, 2001. The next phase describes the releases of Windows Vista on 30 January 2007 and Windows 7 on 22 October 2009. Finally, the latest major spike at the end of 2014 corresponds to the unveiling of Windows 10 in September 2014. However, considering the fact that MS DOS was released in 1981, it makes compilation times before this day look fake or obfuscated intentionally. Next, we indicated how the binary files are distributed according to the architecture they were designed for. It was logical to assume that executables constitute the most popular category, while linked libraries may be supplementary in terms of included malicious API functions as described in the Table 4.16.

To sum up, the dataset represents a real world picture of malware samples distribution and can be considered relevant for both Information Security and Machine Learning areas. However, malware categories can be hard to distinguish since they are too general, and the importance of the numerical features is therefore not high.

Performance Evaluation

The accuracy of the proposed improvements to the NF in multinomial malware detection of a class is compared against the ANN method since both are of the same inherited architecture, while the complexity of the resulting model is different:

Neuro-Fuzzy was implemented in C++ using *Boost* and *Eigen* libraries. The method also includes modifications described by Shalaginov et al. [375]. It uses two steps: SOM grouping and fuzzy rules tuning using single-layer ANN. On the 1^{st} step, the method was run 100 times performing bootstrap aggregation using 1% as suggested in the work by Shalaginov et al. [375] to tackle the generalization problem. The main purpose of NF training was to understand and compare how well the model can actually describe the data it was built from. On the 2^{nd} step, the learning rate of 0.3 was used, while the number of epochs was adjusted.

ANN is available from the *Weka* package and represents multi-output models. It was run with a learning rate equal to 0.3 and momentum 0.0 to eliminate the risk of unstable learning. The Ou et al. [303] used 2-3 layers in Neural Network architecture to test multinomial datasets, yet we concentrate only on 1 and 2 layer architectures, considering 3 layers being more relevant to DNN.

The following metric was used to evaluate the performance of both methods for overall accuracy $Acc = \frac{N_P}{N_S}$, where N_S - number of data samples and N_P - number of properly classified samples according to $max - min$ principle [233] in Mamdani-type rules.

Results Analysis

Once the subsets were generated, we performed a feasibility study of NF application for this task by selecting the most relevant features and evaluating their classification accuracy. Also, some limitations of Neural Network-based architectures were revealed.

Feature selection

To evaluate the utility of manually extracted features for the defined classification tasks, we performed feature selection using *Information Gain* method. Additionally, we evaluated the results of the *RelieFF* and *CFS* methods, and the results look consistent. For Information Gain, we used a threshold equal to 0.1, which resulted in the following feature subsets as described in the Table 4.17.

It can be noticed that selected features have significant relevance for malware families detection, while for malware categories detection, they are too general. Malware analysts may find this work helpful in the identification of specific malware

species based on their attributes. Furthermore, results are given in the Table 4.18.

From the results one can see that NF outperforms ANN, yet gives the same accuracy when increasing the number of training epochs on 2^{nd} stage. For malware families, there can be seen an improvement in the accuracy when using non-linear 2-layers ANN model and a larger number of training epochs. We can say that the improved fuzzy patches [374] are already a good data description and are not affected by training iterations. However, to be more specific, the MAPE (Mean Absolute Percentage Error) metric decreases from 89.64% to 66.02% for simple rectangular patches in NF on a malware families set. For the malware categories dataset, MAPE was reduced from 135.24% to 119.63% for the simple rectangular method, and from 64.38% to 63.13% for the original Kosko method [233]. Yet the proposed earlier method [375] does not have a reduction in MAPE. Furthermore, we investigated the True Positive (TP) and False Positive (FP) rates for each class as is indicated in the Table 4.19. Also, we believe that True Negative and False Negative metrics are less important since we deal with multinomial classification. One can see that among malware families, *hupigon* backdoor has a high detection rate and, among all categories, *trojan* is the most detectable one.

Table 4.11: Description of all 37 numerical features that were extracted from raw Windows PE32 malware characteristics

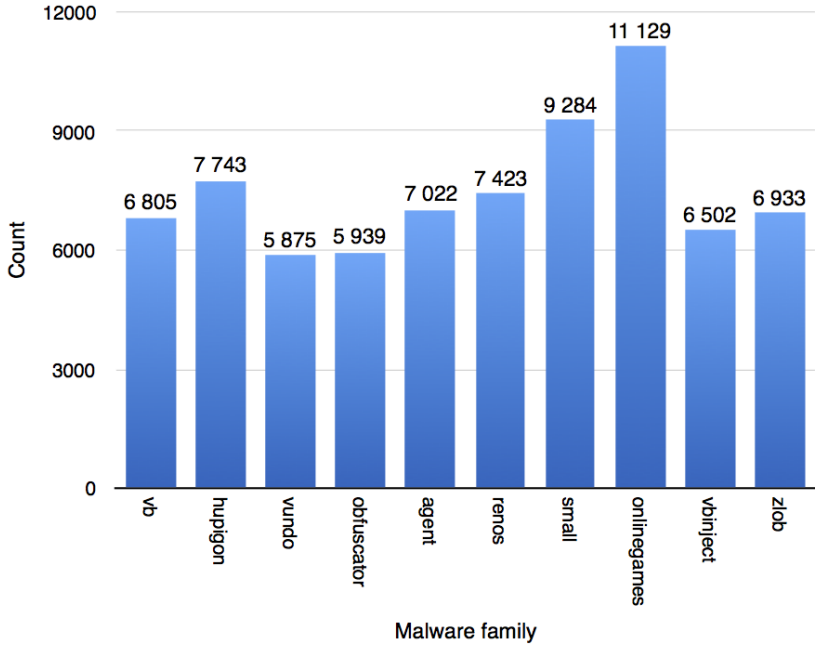
From PEframe output	
pe_api	The number of suspicious API class
pe_debug	The number of opcodes recognized as common anti debug techniques
pe_packer	The number of packers discovered by PEframe
pe_library	The number of DLL calls in the file. Retrieved from PEframe
pe_autogen	The number of autogens discovered by PEframe
pe_object	The number of object calls discovered by PEframe
pe_executable	The number of .exe calls within the file
pe_text	The length of the 'text'-field
pe_binary	The number of binary(.bin) files called by the file
pe_temporary	The number of .tmp files accessed by the file
pe_database	The number of .db files accessed by the file
pe_log	The number of log entries accessed by the file
pe_webpage	The number of web pages access by the file
pe_backup	The number of backups the file performs or accesses
pe_cabinet	The number of references to .cab files
pe_data	The number of .dat files accessed by file
pe_registry	The number of registry keys accessed or modified by the file
pe_directories	The number of directories accessed by the file
pe_dll	The number of DLL's accessed by the file
pe_detected	The number of suspicious sections in the file
From VirusTotal report	
vt_codesize	The size of the code in the file, retrieved from virustotal
vt_res_langs	The number of resource languages detected in the file
vt_res_types	The number of PE32 resource types detected in the file
vt_sections	The number of PE32 sections in the file.
vt_entry_point	Decimal value of entry point, i.e. the location in code where control is transferred from the host OS to the file.
vt_initDataSize	The size of the initialized data.
vt_productName	The length of the field 'ProductName'. Can e.g. be "Microsoft(R) Windows(R) Operating System"
vt_originalFileName	The length of the original file name
vt_unitializedDataSize	The size of the part of the file that contain all global, uninitialized variables or variables initialized to zero.
vt_legalCopyright	The length of the field LegalCopyRight. E.g."(C) Microsoft Corporation. All rights reserved."
From other Linux command line tools	
size_TEXT	The first output from the 'file' command. This is the size of the instructions
size_DATA	The second output with size of all declared/initialized variables
size_OBJ	The third output that contains the size of all unitialized data, i.e the BSS-field
size_TOT	The fourth output that indicates the sum of the text, data and bss fields of the file
filesize	The total size of the file, including metadata

Table 4.12: 35 most frequent malware categories and families found among Windows PE32 files

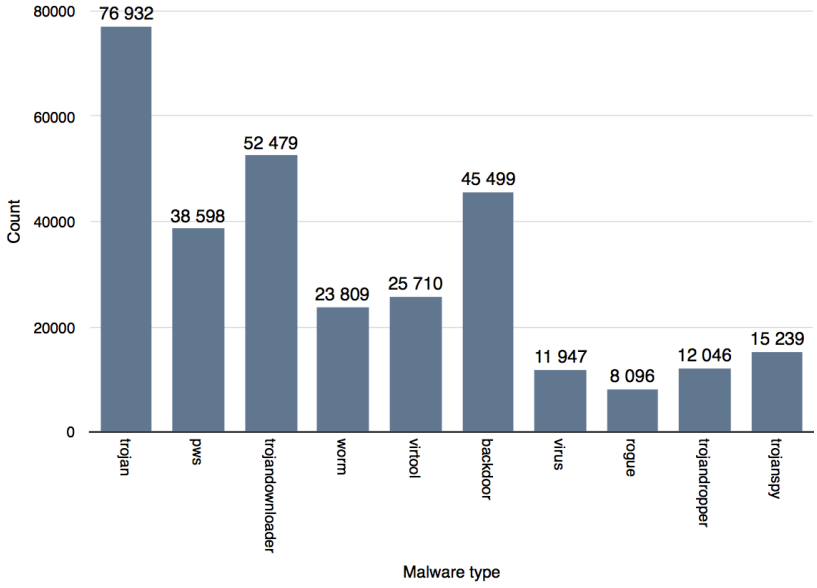
Top Malware families		**Top Malware categories**	
Label	Number	Label	Number
onlinegames	11,129	trojan	76,932
small	9,284	trojandownloader	52,479
hupigon	7,743	backdoor	45,499
renos	7,423	pws	38,598
agent	7,022	virtool	25,710
zlob	6,933	worm	23,809
vb	6,805	trojanspy	15,239
vbinject	6,502	trojandropper	12,046
obfuscator	5,939	virus	11,947
vundo	5,875	rogue	8,096
frethog	5,593	adware	5,294
delf	4,833	browsermodifier	3,051
winwebsec	4,273	trojanclicker	1,962
farfli	4,137	trojanproxy	1,502
bifrose	4,068	spammer	886
zbot	3,941	dialer	806
c2lop	3,892	monitoringtool	806
alureon	3,702	hacktool	730
delfinject	3,631	ransom	671
startpage	3,553	exploit	566
bancos	3,406	ddos	491
banload	3,399	program	362
vobfus	3,319	constructor	262
lolyda	2,841	dos	177
injector	2,646	spyware	169
gamania	2,636	joke	98
tibs	2,378	settingsmodifier	70
taterf	2,341	softwarebundler	52
allaple	2,283	trojannotifier	7
lmir	2,254	tool	4
ircbot	2,160	spoofer	4
banker	2,137	flooder	3
hotbar	2,079	remoteaccess	3
bho	2,065	nuker	3
poison	2,039	misleading	3

Table 4.13: Number of selected features out of 27 initial features for each of the method

Dataset	**Feature Selection methods**		
	Cfs	**InfoGain**	**ReliefF**
10 families	15	15	25
100 families	16	17	27
500 families	16	13	27
10 types	12	11	23
35 types (full)	14	15	25



(a) 10 most frequent families



(b) 10 most frequent types

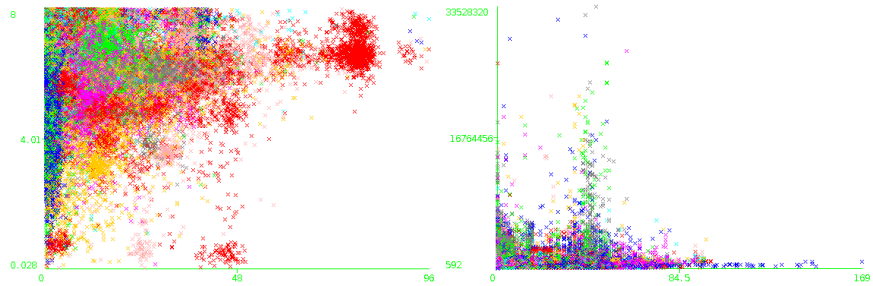
Figure 4.8: Distribution in malware families and types datasets

Table 4.14: Commonly selected features for malware families and types datasets using different feature selection methods

Malware families			**Malware types**	
10 families	1000 families	500 families	10 types	35 types
vt_res_langs	vt_codesize	vt_codesize	**vt_codesize**	**vt_codesize**
vt_res_types	**vt_res_langs**	**vt_res_langs**	**vt_entry_point**	**vt_entry_point**
vt_sections	**vt_sections**	**vt_sections**	**vt_initDataSize**	**vt_initDataSize**
vt_entry_point	**vt_entry_point**	**vt_entry_point**	**vt_productName**	**vt_productName**
vt_initDataSize	**vt_initDataSize**	**vt_initDataSize**	**vt_unitializedDataSize**	**vt_unitializedDataSize**
vt_productName	**vt_productName**	**vt_productName**	**vt_legalCopyright**	**vt_legalCopyright**
vt_originalFileName	**pe_api**	**pe_api**	**pe_dll**	pe_api
vt_unitializedDataSize	pe_packer	pe_packer	**size_TEXT**	**pe_dll**
pe_api	pe_library	pe_library	**size_DATA**	**size_TEXT**
pe_debug	pe_dll	pe_executable	size_OBJ	size_DATA
pe_dll	size_TEXT	**pe_dll**	**filesize**	**size_OBJ**
size_DATA	**size_DATA**	size_TEXT		size_TOT
filesize	**filesize**	**size_DATA**		**filesize**
	entropy	**filesize**		
		entropy		

Table 4.15: Accuracy of Soft Computing and selected Hard Computing methods, in %

Dataset	**Features**	**Soft Computing**				**Hard Computing**	
		Naive Bayes	**Bayes Net.**	**MLP**	**SVM**	**C4.5**	**Rand.Forest**
10 f.	Full set	23.9408	**72.6462**	42.3133	32.9824	83.2871	88.7965
	Cfs	20.8760	70.8285	40.7206	32.7962	83.9180	88.1990
	InfoGain	23.9354	**72.6462**	**51.5947**	38.0952	**84.2100**	**88.9572**
	ReliefF	**31.7835**	65.9996	41.0703	**41.2283**	82.7821	87.2279
100 f.	Full set	20.9062	61.6056	11.8116	–	76.6250	81.3078
	Cfs	21.0008	**61.6851**	**18.1451**	–	77.8019	**81.9535**
	InfoGain	20.1905	60.9452	15.9034	–	**77.8204**	81.8329
	ReliefF	**23.9406**	57.3495	17.4743	–	75.3771	79.4565
500 f.	Full set	16.1718	**57.2137**	9.1806	–	72.4210	–
	Cfs	**16.7610**	56.8853	**11.5360**	–	**73.2847**	–
	InfoGain	13.7654	54.1846	6.8251	–	72.4487	–
	ReliefF	2.8466	52.2410	11.3481	–	70.7730	–
10 t.	Full set	**10.4316**	**51.3618**	**28.4829**	–	73.6789	**78.6000**
	Cfs	7.3857	51.1862	25.9661	–	**73.8200**	77.1639
	InfoGain	4.9063	49.9312	26.9623	–	73.6099	77.5177
	ReliefF	9.9650	47.2420	27.5317	–	72.4954	77.0888
35 t. (full)	Full set	2.3576	48.4487	27.2826	–	72.6272	77.8797
	Cfs	1.9894	**48.6290**	25.1683	–	73.7480	77.3324
	InfoGain	1.9785	48.4140	25.7842	–	**73.8254**	**77.9687**
	ReliefF	**3.3517**	44.9438	**28.0675**	–	71.7842	76.4943



(a) 10 fam.: ‘pe_api‘ vs ‘entropy‘ **(b)** 10 cat.: ‘pe_api‘ vs ‘filesize‘

Figure 4.9: Distribution of samples in families and categories datasets using different static features for 10 classes

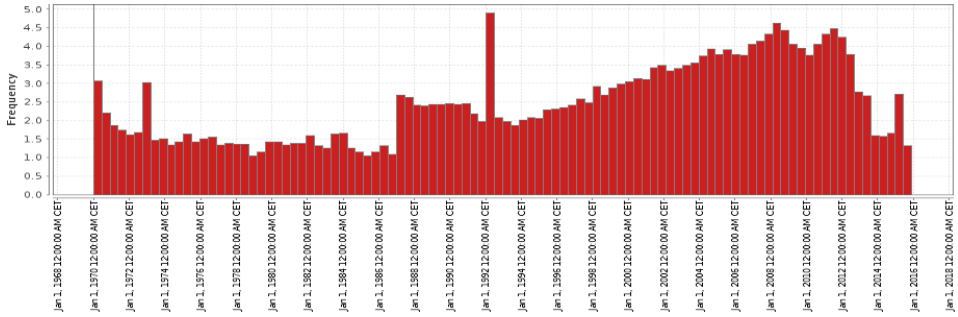


Figure 4.10: Log-scaled plot of the malware compilation time frequency built with a help of RapidMiner [14]

Table 4.16: Most popular PE32 architectures found in the dataset according to Linux ‘file‘ command

Architecture	Samples
PE32 executable (GUI) Intel 80386, for MS Windows	231,445
PE32 executable (DLL) (GUI) Intel 80386, for MS Windows	57,012
PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed	50,608
PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft Installer self-extracting archive	11,009
PE32 executable (GUI) Intel 80386, for MS Windows, PECompact2 compressed	9,349

Table 4.17: Selected features for malware families and malware categories datasets using Information Gain

10 Malware families		**10 Malware categories**	
Info Gain	Feature name	Info Gain	Feature name
1.2678	vt_entry_point	0.5655	vt_entry_point
0.9768	size_TOT	0.5042	size_TEXT
0.9627	size_TEXT	0.4771	size_TOT
0.9559	filesize	0.4457	size_DATA
0.9518	vt_initDataSize	0.4250	filesize
0.8712	size_DATA	0.4043	vt_initDataSize
0.8572	vt_codesize	0.3866	vt_codesize
0.6456	pe_api	0.1893	entropy
0.5529	entropy	0.1725	pe_api
0.4325	vt_unitializedDataSize	0.1697	vt_unitializedDataSize
0.3929	vt_productName		
0.3895	vt_sections		
0.3622	vt_res_langs		
0.3398	pe_library		
0.3197	vt_originalFileName		
0.2814	vt_res_types		
0.2741	pe_debug		
0.2530	pe_dll		
0.2506	pe_packer		
0.2230	vt_legalCopyright		

Table 4.18: Overall classification accuracy of the Neuro-Fuzzy methods and ANN (with 1 and 2 hidden layers), in %

Method	**Epochs in ANN / 2^{nd} NF step**		
	10	50	100
Dataset: Malware families			
NF (10 rules)	39.6196	39.6196	39.6196
$\text{ANN}_{1\ layer}$	22.2262	22.3347	22.5102
$\text{ANN}_{2\ layers}$	32.0742	31.8813	32.6422
Dataset: Malware categories			
NF (27 rules)	26.4665	26.4665	26.4665
$\text{ANN}_{1\ layer}$	24.7884	24.7884	24.7884
$\text{ANN}_{2\ layers}$	25.0774	24.9050	24.9198

Table 4.19: True Positive and False Positive rates of Neuro-Fuzzy for 10 malware families and 10 malware categories

Family	vb	hupigon	vundo	obfuscator	agent	renos	small	onlinegames	vbinject	zlob
samples	6,805	7,743	5,875	5,939	7,022	7,423	9,284	11,129	6,502	6,933
TP rate	0.3595	**0.8080**	0.5405	0.1222	0.1633	0.3276	0.5229	0.6084	0.2076	0.4295
FP rate	0.0226	0.2033	0.0233	0.1185	0.0341	0.0101	0.1397	0.0303	0.0261	0.0262
Category	trojan	pws	trojandownloader	worm	virtool	backdoor	virus	rogue	trojandropper	trojanspy
samples	76,932	38,598	52,479	23,809	25,710	45,499	11,947	8,096	12,046	15,239
TP rate	**0.6084**	0.1954	0.1385	0.1608	0.2112	0.3392	0.0857	0.0000	0.0744	0.0769
FP rate	0.5220	0.0432	0.0517	0.0097	0.0614	0.1528	0.0098	0.0000	0.0193	0.0152

It can be seen that with a growing amount of samples, the accuracy of the NF method may not be influenced by simply tuning the number of training epochs, as can be seen from the categories dataset. Additionally, to cross-validate, we applied Random Tree methods on both datasets. The achieved accuracy was 83.567% for malware families using the size of the tree equal to 23,755 and 73.3402% for malware categories with a tree size of 151,743. We can state that such a model has enormous complexity and is hardly-applicable in real life problems that require good generalization. Also, it means that the extracted feature may not work well for the categories dataset as we can see from the values of Information Gain in the Table 4.17. Thus, better quality features must be analytically defined.

Summary: We explored the application of NF for multinomial classification of malware families and categories. We collected a novel dataset consisting of 400k samples for static malware analysis. In the literature, there are only a few works that consider multinomial malware detection, while others only target binary problems. The proposed is a way to enhance the accuracy and generalization of NF to be able to handle such complex and large-scale problems. NF performs well considering the complexity of the problem and non-linearity of the data. However, there is an identifiable limitation in the method despite the proposed improvements. Also, it archived better accuracy than non-linear ANN. This shows that NF can handle such data, though the quality and utility of the extracted features have a crucial influence on the method's performance, which requires additional analysis of the PE32 headers.

4.2.4 Dynamic Behavioural Analysis

Malware developers have been employing more and more advanced techniques in their software to remain unnoticed for as long as possible, and to cause as much harm as possible. They can use fake Windows certificates, zero-day vulnerabilities and default software settings, etc. as described by Wu et al. in 2016 [444], making it difficult to notice abnormalities. Furthermore, a set of obfuscation methods is often applied such as encryption, polymorphism, metamorphism, dead code insertions, or instruction substitution [356] to conceal the real functionality logic of the software. In addition to this, MS Windows is a known target of many attacks crafted by famous viruses such as Stuxnet, Duqu and Flame [76]. Multiple market share surveys suggest that more than 50% of desktop computers and laptops users utilize MS Windows as an Operating System (OS) [35]. At the same time, nearly 10% of the users still have Windows XP installed, which is no longer a supported OS version [34].

Experimental Design & Methodology

Our main goal is to create a light-weight approach that does not require a specifically-tuned sandbox nor major changes to the desktop platform, and can deliver a human-understandable multinomial classification model to the end customer. The idea is to utilize different behavioural characteristics which are commonly observed in malware analysis [333, 252, 98]. Further in this thesis, we follow a known dynamic malware analysis cycle described by Ligh et al. in the Malware Cookbook [251]. Its seven steps are shown in the Figure 4.11.

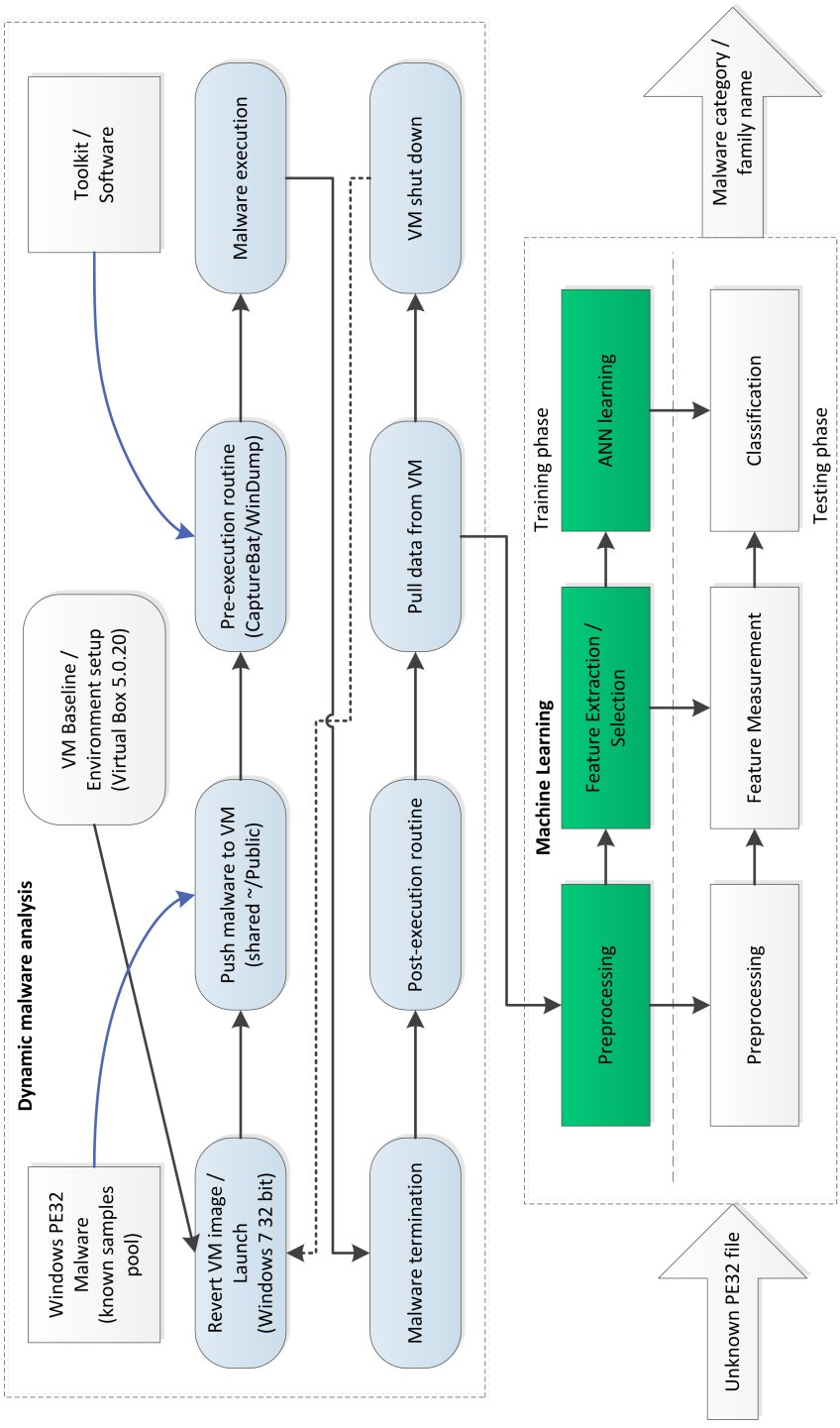


Figure 4.11: Dynamic malware analysis [251]

Furthermore, having information gained from a PE32 header, files with *GUI* were selected. The reason for this filtering is that DLLs might be difficult to evaluate without invoking API calls. This brings significant problems to automated dynamic analysis since it can produce very small amounts of data or require an unreasonably long waiting time. Finally, we also filtered out executables that contain anti-debug and anti-VM features according to PEframe. Afterwards, files were filtered by presence in each of the categories and families were selected for preliminary analysis.

This ongoing study is aimed at exploring both effective automated malware detection and the limitations of ML methods. Therefore, we used 1,000 samples of the 10 most frequent malware families and 1,000 samples of the most frequent malware categories. This is a fraction of the collected dataset, and used mainly as a proof-of-concept demonstration due to the significant amount of time required for the malware execution and logs processing.

Behavioural analysis as a malicious indicator

Multiple technical reports by famous security labs such as Kaspersky, Symantec, etc. investigate particular malware samples that belong either to a family or category. This process is often manual and involves the study of multiple aspects, also during execution of the malware. We believe that behavioural analysis may speed up this process and enable similarity-based deduction of zero-day exploits. Many researchers address behavioural characteristics that showed their efficiency for malware detection [134, 158, 201], including the case of multinomial detection described by Rieck et al. [334]. Survey by Gandotra et al. [158] shows that quite a few researchers prefer to use behavioural characteristics when classifying malware into malware families using ML.

Behaviours Characteristics: Disk activities

Most of the software leaves disk patterns when launched. It can be either some meta data required for proper operation or user-specific sensitive information. Using such logic, we can make a hypothesis that a specific malware category might be characterized by its own disk activity patterns. We divide disk activities into two sub-domains:

- *Low-level access by the application* that includes modification, deletion and writing to the file on a disk storage accessible for user as presented by Lin [252] earlier. Also, if the launched executable was immediately deleted form the disk, it might indicate a desire to hide its presence. Similarly, Cheng et al. [98] monitored *FileWrite* API calls to find malicious indicators. We can see such operations in the operation of the *trojanspy* category

malware. For example, if we execute *TrojanSpy:Win32/YBad.B* (md5 sum e4c36489ca8f4d11a77f9d322a5b20d8) sample, the disk activities are going to be as described in the Figure 4.12. It can be also see from the VirusTotal reports that malware creates specific DLL and EXE files into System32 directory to be able to run itself automatically and look for login windows to attach to. This malware was designed to steal user-sensitive data on the victim's computer.

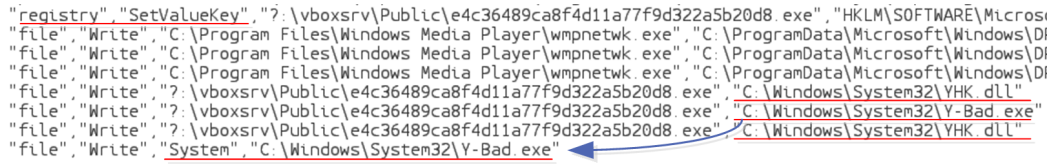


Figure 4.12: Trojan Spy creates files in System32 directory

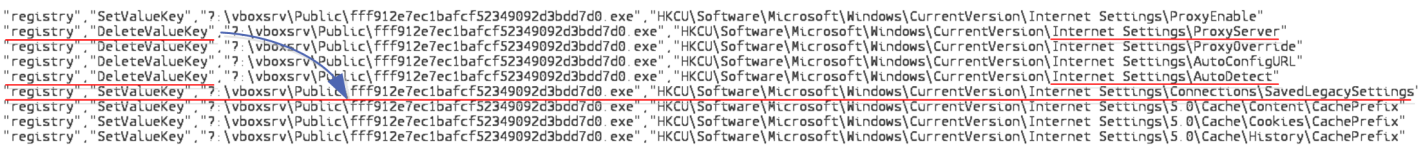


Figure 4.13: Trojan Dropper activity that makes modification in Windows registry

- *Behaviours Characteristics: Registry patterns* describes changes into the configuration database of OS. Windows registry is a complex database that is used to store low level system and application settings according to the MS manual page [277]. Because of hierarchical storage, this must be considered separately from disk activities rather than just as a modification of the corresponding files in the *%SystemRoot%\System32\Config* directory. For example, looking at the logs from *TrojanDropper:Win32/VB.AU* execution (md5 sum fff912e7ec1bafcf52349092d3bdd7d0) that are given in the Figure 4.13. One can see that the malware makes changes in the MS Internet Explorer security zones and privacy settings. In particular, it disables proxies from preventing the use of the Internet and updating anti-virus software, since most the environments use some kind of built-in proxies.

  Further, Dolan-Gavitt [124] performed a forensics analysis of the Windows registry in memory and provided guidelines on how it can be extracted. Rieck et al. [333] used the creation or modification of registry keys as one of the main features in multinomial malware classification. It can be seen that the amount of data stored in the registry is huge and may lead to identification of the malicious activities as shown in the research by Carvey [88]. In this paper, we limit the number of data to be examined within the registry [277]. We will look into local computer settings (HKEY_LOCAL_MACHINE or HKLM) that are further divided into the following keys: SAM, SECURITY, SYSTEM and SOFTWARE. User settings HKEY_USERS (HKU) are also within our scope, and HKEY_CURRENT_USER (HKCU) is of a particular interest. Finally, HKEY_CLASSES_ROOT (HKCR) may indicate any settings changes by malware being launched.

So, one can say that files and registry alterations may be considered as lead disk indicators of malicious activities during file execution. However, some of the malware categories do not leave any disk or registry traces, making such behavioural characteristics less efficient in automated classification.

Behaviours Characteristics: Network traffic

Network activity is one of the most important requirements for successful malware operation. It communicates with an attacker, downloads tasks or sends sensitive user data. Out of the mentioned earlier relevant works, only Rieck et al. [333] briefly investigates IRC connections and ping scans. The number of artifacts that can be extracted from network traffic dumps however are much larger. Some simple examples of such artifacts will be shown below. The Figure 4.14 shows traces left by *TrojanDownloader:Win32/Agent.HA* (md5 sum 3ed12bffaa3bd840c1895e8bd87c4fc1) as it was iteratively trying to get a *calc.exe* file without success. Attempts to down-

load system-like executables are a strong indicator of suspicious activity.

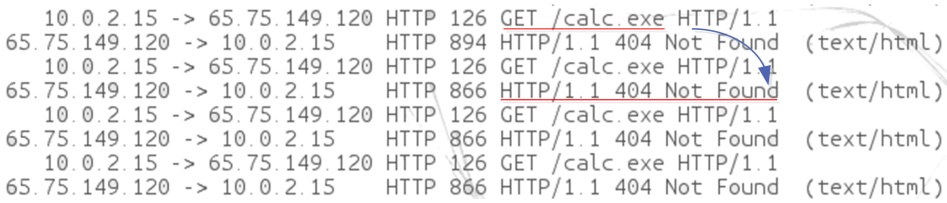


Figure 4.14: Trojan Downloader attempts to retrieve an executable

Furthermore, we can see how the *Backdoor:Win32/Hostil.gen!A* (the md5 sum of the malware file is 51b19c82c0b9c8e6fdda7875e3266ad7) sends sensitive information (IP address) and some encrypted parameters out to an advertising web-site through HTTP GET request as shown in the Figure 4.15.

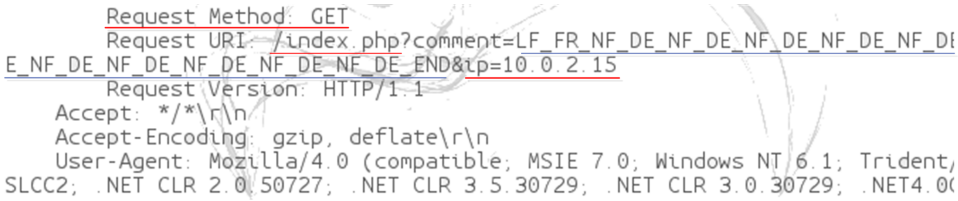


Figure 4.15: Backdoor sends encoded GET request with IP address

Behaviours Characteristics: Memory footprints

From the literature study, we can see that memory footprints are much harder to observe than disk or network traffic patterns. Such analysis required either (i) extensive knowledge about the values of the memory space since an attacker's memory alteration may remain unnoticed. Prakash et al. [318] claimed that memory analysis without ground-truth could not be very trustworthy, especially on proprietary OS (such that MS Windows). They investigate the accuracy and efficiency of traversal-based and signature-based memory analysis tools (Volatility [32] framework and its plug-ins), which are designed to provide information about processes, modules, files etc. Alternatively, memory analysis needs (ii) specifically-crafted software and detection schemes. The accuracy and efficiency of robust field-based signature schemes are described by Dolan-Gavitt et al. [125]. Another work showed a graph-based brute-force approach SigGraph by Lin et al. [254]. They compare results from binary analysis tools and Volatility over the Virtual Machine memory. The authors claimed that the achieved results of traversal-based and signature-based methods showed reduced accuracy, making memory patterns

analysis is out of the scope of this paper since the characteristic collection process is cumbersome and hardly yields acceptable results, especially when using automated classification.

Performance Evaluation

The experimental setup and corresponding tools are given below. We refer to the automated dynamic malware analysis cycle suggested by Ligh et al. [251] and sketched in the methodology section 4.2.4 above. Finally, performance metrics and achieved results are presented.

For our experiments, we used ANN, a famous Soft Computing and ML method also described by Kononenko et al. [232]. This method has great generalization capability and the ability to learn from complex non-linear relationships in data without much prior knowledge [367, 418]. ANN requires almost no manual model parameters tuning, since it is done automatically. Additionally, it offers highly non-linear modeling, which is important in multinomial classification in contrast to binary classification, where a number of methods exist to make a linearly separable decision model. In our experiments, we use multilayer ANN, which can also be considered as Deep Neural Network, where the number of hidden layers is greater than 3. ANN consists of input neurons with corresponding weights w_j for each of the input data x_j, the number of features N and specifically-designed activation function $g()$. The output y_i of the ANN is as follows: $y_i = g(\sum_{j=1}^{N} w_j \cdot x_j)$, and is designed to perform an optimization (minimization) of the following cost (error) function $E = \frac{1}{2} \cdot (d_i - y_i)^2$, where d_i - is the original label of the data sample. We used 500 training epochs while learning from the data. The number of hidden layers was calculated using a "rule of thumb": (number of features + number of classes) / 2. Finally, a 5-fold cross validation was applied for performance estimation.

Multinomial classification is a problem investigated in this thesis and therefore it is not sufficient to simply use overall accuracy as the ultimate performance metric. Such a metric might be suitable for a balanced binary classification problem, where it gives a clear picture of the properly classified samples. When it comes to multinomial classification however, we need a specific per-class performance metric, which can show the number of properly classified samples in a particular class. Therefore, we use ML community-accepted metrics according to Kononenko et al. [232], such as True Positive Rate (TPR, also called as Sensitivity or Recall): $TPR = TP/P$, which defines the number of properly classified samples of a particular class TP with respect to the original number of samples in this class P. Another metric is False Positive Rate (FPR) rates of each class for this purpose: $FPR = FP/N$, which defines the fraction of samples of other classes

that are classified as the current class FP with respect to the number of all other samples from other classes excepting the current class N. Also, we believe that True Negative and False Negative metrics are less important since we deal with multinomial classification.

Results & Analysis

Both datasets (families and categories) contain 49 features that can be extracted automatically: 30 disk- and log-related, the checking of whether the exe file was deleted, and 18 network-related as shown in the Table 4.20.

Table 4.20: Overview of the constructed features describing dynamic behaviour

Disk- and log-related features			Network-related features	
memory_processCreated	disk_registryHKU	malware_disk_registrySetValueKey	network_fileSize	network_tcp
memory_processTerminated	disk_registryHKLM_SAM	malware_disk_registryDeleteValueKey	network_numPackets	network_icmp
disk_numberRecords	disk_registryHKLM_SECURITY	malware_disk_registryHKLM	network_dataSize	network_http
disk_fileWrite	disk_registryHKLM_SYSTEM	malware_disk_registryHKCU	network_dataRate	network_httpGET
disk_fileDelete	disk_registryHKLM_SOFTWARE	malware_disk_registryHKCR	network_avgPacketSize	network_httpGETexe
disk_registrySetValueKey	disk_fileNamePresent	malware_disk_registryHKU	network_avgPacketRate	network_httpGETtxt
disk_registryDeleteValueKey	malware_memory_processCrtd	malware_disk_registryHKLM_SAM	network_uniqueProtocols	network_httpGETphp
disk_registryHKLM	malware_memory_processTermntd	malware_disk_registryHKLM_SECURITY	network_uniquePacketSzs	network_httpGETjpg
disk_registryHKCU	malware_disk_fileWrite	malware_disk_registryHKLM_SYSTEM	network_uniqueIPs	
disk_registryHKCR	malware_disk_fileDelete	malware_disk_registryHKLM_SOFTWARE	network_udp	

Malware families TP rate and FP rate are presented in the Table 4.21. By using the feature selection method Information Gain [232] we found that the largest merit in differentiation between families have (1) registry "*SetValueKey*" operations by executed files and (2) size of the data transferred via a network during observation. Class *Onlinegames* can be detected with a high degree of confidence using disk features, while class *vundo* is more distinguishable using network characteristics.

Malware categories classification results gave interesting results as shown in the Table 4.22, considering the variety of malware families in each category. According to InfoGain, (1) the registry "*SetValueKey*" plays a key role as a disk feature in families separation, while (2) the number of a unique network packet's sizes has the biggest merit when making a separation between categories. Class *Virtool* shows the largest performance on disk features and is a hacker tool with a combination of trojan and obfuscation methods to hide its presence and make a victim's computer into a zombie-machine [33]. One can see that the results of the dynamic behavioural classification outperform static analysis in the detection of some classes [381]. In particular, our method is able to detect *Rogue*, a fake anti-virus software, which was not detected by static analysis at all.

Table 4.21: Classification performance of ANN on 10 malware families

ANN layers	Features	Performance	Classes									
			agent	hupigon	obfuscator	onlinegames	renos	small	vb	vbinject	vundo	zlob
21	Disk	TP rate	0.070	0.420	0.200	**0.930**	0.020	0.410	0.110	0.390	0.050	0.520
		FP rate	0.011	0.007	0.140	0.000	0.001	0.073	0.021	0.247	0.008	0.257
14	Network	TP rate	0.010	0.190	0.230	0.200	0.420	0.370	0.060	0.210	**0.490**	0.320
		FP rate	0.010	0.094	0.024	0.063	0.101	0.067	0.053	0.102	0.113	0.204
30	Both	TP rate	0.050	0.420	0.230	**0.930**	0.450	0.360	0.130	0.330	0.510	0.520
		FP rate	0.026	0.007	0.019	0.003	0.103	0.061	0.067	0.111	0.120	0.158

Table 4.22: Classification performance of ANN on 10 malware categories

ANN layers	Features	Performance	Classes									
			backdoor	pws	rogue	trojan	trojandownloader	trojandropper	trojanspy	virtool	virus	worm
21	Disk	TP rate	0.040	0.320	0.180	0.000	0.130	0.050	0.500	**0.560**	0.050	0.190
		FP rate	0.002	0.060	0.136	0.000	0.030	0.022	0.008	0.463	0.002	0.163
14	Network	TP rate	0.050	0.350	**0.530**	0.070	0.390	0.170	0.190	0.070	0.070	0.050
		FP rate	0.017	0.206	0.192	0.022	0.077	0.106	0.144	0.032	0.034	0.066
30	Both	TP rate	0.060	0.280	0.480	0.120	0.220	0.190	**0.570**	0.160	0.070	0.410
		FP rate	0.016	0.044	0.196	0.073	0.057	0.091	0.027	0.094	0.017	0.212

Summary: There has been a great need for malware categorization due to the recent emergence of new ways to attack computer systems. Existing literature distinguishes between general *malware categories* and more specific *malware families*. Previous relevant works on multiple categories considered only a few of them based on outdated and imbalanced datasets. Based on the literature review we found that multinomial detection has been neither sufficiently explored nor tested on recent malware samples. Moreover, authors of similar research used only general characteristics in specifically-designed sandboxes. In this thesis, we use a labelled collection of malware categories and families from the end of 2015. More granular disk-, log-, and network-based behavioural characteristics were extracted without use of specific sandboxes. This is an ongoing work and we are seeking proof that such behavioural characteristics can be used for ML-aided automated malware classification by Artificial Neural Networks. Today, this is a prominent method already successfully used in malware detection. Furthermore, the application of Soft Computing may help boost accuracy by better modelling non-linear feature dependencies. We believe that the methodology used in this study contributes to supporting the decision of a malware analyst, and provides better protection against unknown malware samples that may appear in a corporate environment.

4.3 Intrusion Detection[4]

This thesis is also aimed at Intrusion and focused on analysis of network traffic for gathering evidences of attacks or relevant intrusion information with a scope of prevention or future testimony. Our scope is primary focused on firewall rules generation to indicate the attacks coming from outside, while using Mamdani-type classification rules. NF showed great performance in malware detection tasks as studied by Singh et al. [394]. Another important area of ongoing research is understanding and spotting attack patterns in network traffic in order to be able to prevent network compromise and to investigate incidents that have already occurred. Shanmugavadivu et al. [386] presented a work that studied the applicability of fuzzy logic for the intrusion detection dataset from the obsolete *KDD Cup 1999* challenge. However, despite the successful application of the fuzzy logic for digital forensics challenges, the main concern is with large-scale data that can result in a hardly-explainable model without manual analysis. One of the well-known NF architectures was proposed by Kosko et al. in 1997 [233]. It uses Self-Organizing Maps (SOM) to extract parameters of fuzzy rules automatically without human interaction. It works well on small-scale datasets with tens of thousands of samples.

[4]The main ideas of this section are published under the contributions [375, 378]

4.3.1 Datasets

The 10% of **KDD Cup 1999** dataset is publicly available and can be accessed using the UCI Machine Learning Repository [250]. It is a well-known dataset that was released by Kibler et al. [107] in 2000 as a part of KDD Cup 1999 challenge. It is a set of 41 characteristics of network packets characterized "bad" or "good". The task is from Intrusion Detection and the model derived from the data can be used to flag any malicious activity. The features values are real, symbolic, binary, and integers. The Feature Selection was performed based on the method proposed by Nguyen et al. [294] in 2010 with 9 features that contribute most to the classification of the benign and malicious connections: 5 - *src_bytes*, 6 - *dst_bytes*, 10 - *hot*, 12 - *logged_in*, 14 - *root_shell*, 22 - *is_guest_login*, 29 - *same_srv_rate*, 37 - *dst_host_srv_diff_host_rate*, 41 - *dst_host_srv_rerror_rate*. According to the authors, these features provide the classification without much loss of accuracy in comparison to the original dataset. The dataset includes 97,278 samples of the 1^{st} class ("normal") and 396,743 samples of the 2^{nd} class ("attacks") out of 494,022 samples in total.

Critics of KDD CUP 1999 dataset

We are aware of the fact that the KDD CUP 1999 dataset is nearly 18 years old. During this period of time, the network infrastructure developed all the way from Fast Ethernet 100Mbps in 1995 to 100th of Gbps in 2017. Dozens of new types of applications have been developed, resulting in an exponential growth of vulnerabilities and possible attack vectors. In 2000, McHugh performed a critical analysis of DARPA 1998 and 1999 datasets [274] and found that there exist a number of shortcomings. Similarly, Tavalaee et al. [412] performed a thorough investigation of KDD CUP 1999 and concluded that the dataset posses several intrinsic problems related to packets distribution. The authors found that 78% of the records in the training dataset are duplicates. This means that the number of distinct data records in the dataset is actually much smaller. Tavalaee et al. proposed an alternative dataset called NSL-KDD. This new dataset includes 125,973 trained data samples and 22,544 tested. Considering this fact, we have decided to proceed with the KDD CUP 1999 dataset for our experiments due to the fact that one of the goals of this research is to test the new Soft Computing method on large-scale datasets. To the authors' knowledge, there is no dataset except KDD CUP 1999 that describes network attacks and is also large-scale.

4.3.2 Experimental Design

In this thesis, we will study the applicability and required improvement of the original Kosko method for large-scale data analysis, including network traffic dumps

with several hundreds of thousands of samples. Moreover, the comparative study of Hard Computing and Soft Computing methods was made to understand the importance of the proposed improvements. Finally, we make an estimation of the efficiency of the generated model by analysing the computational time required to learn and inference. Moreover, space complexity is low, which makes it feasible to also apply the model on embedded devices with limited storage capacity.

4.3.3 Performance Metrics

A special focus was put on the accuracy, model complexity, and time required to learn and infer from the model. As for performance evaluation, we used classification and regression metrics. Classification estimated the number of properly classified samples, while the regression metrics included the following measures: Mean Absolute Error (MAE), Relative Absolute Error (RAE), and Mean Absolute Percent Error (MAPE) from the value of the center of gravity defuzzifier.

$$MAE = \frac{1}{N_S}\sum_{i=1}^{N_S}|y_i - d_i|; \tag{4.1}$$

$$MAPE = \frac{1}{N_S}\sum_{i=1}^{N_S}|\frac{y_i - d_i}{d_i}| \cdot 100\% \tag{4.2}$$

$$RAE = \frac{\sum_{i=1}^{N_C}|y_i - d_i|}{\sum_{i=1}^{N_S}|d_i - \bar{d}|}; \tag{4.3}$$

$$Acc = \frac{N_P}{N_S} \tag{4.4}$$

where y_i - the output of the defuzzifier on the 2^{nd} stage of NF, d_i - the actual class of the sample, $\bar{d}$ - average value of actual class, and N_P - number of properly classified samples according to $max-min$ inference principle [233] in Mamdani-type rules. We refer to the $Acc, \%$ when taking about the classification accuracy.

4.3.4 Results & Analysis

In the Chapter 1, we mentioned four hypotheses that could facilitate the forensically-sound analysis of large-scale datasets. First, we hypothesize that the lower number of nodes in SOM (the 1^{st} step of NF) does not necessitate a significant drop in accuracy. The results for the earlier-mentioned performance metrics are presented in the Table 4.23. The Vesanto method has lower accuracy on rectangular patches due to significant overfitting caused by using a greater number of specific rules. Bagging may help to generalize the model when using a lower amount of more

general rules.

Table 4.23: Performance comparison (regression, classification) of the proposed improvements

SOM size	MF	Full dataset				Bootstrap aggregation			
		MAE	**RAE**	**MAPE**	**Acc**,%	**MAE**	**RAE**	**MAPE**,%	**Acc**,%
Vesanto	Simple	0.168	0.532	14.728	38.272	0.213	0.674	20.236	19.847
	Kosko	0.817	2.583	41.620	80.080	0.103	0.325	9.915	77.277
	Proposed	0.579	1.832	29.195	**98.790**	0.040	0.126	3.590	**99.058**
		10,231 rules				**211** rules			
Proposed	Simple	1.124	3.556	66.020	14.059	0.198	0.627	19.425	73.135
	Kosko	0.454	1.437	26.914	91.710	0.205	0.650	10.896	90.455
	Proposed	0.077	0.244	6.029	**94.571**	0.118	0.374	5.970	**98.787**
		50 rules				**39** rules			

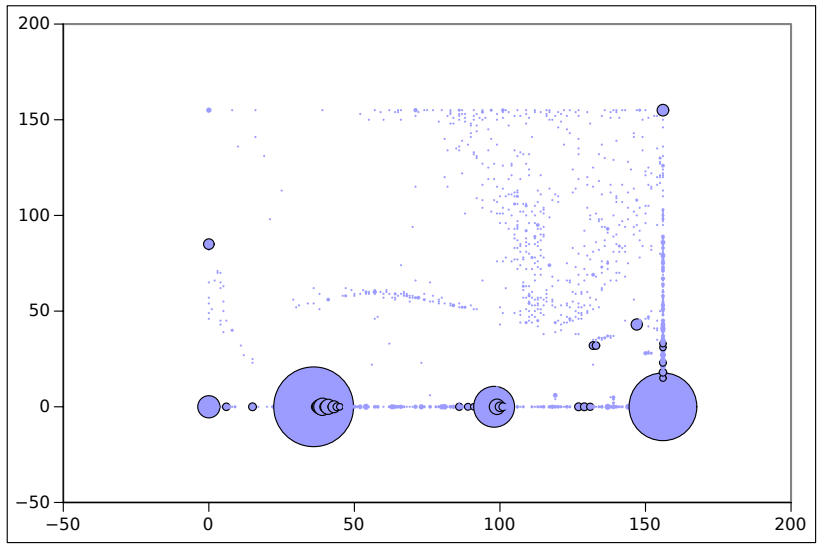
Moreover, the results of SOM clustering are shown in the Figure 4.16, where the bubble plots depict the samples distribution for 1% of the dataset using Vesanto method and proposed methods of an optimal SOM size determination. From the left part of the Figure 4.16, we notice that not all of the nodes are filled with samples, which are empty. Moreover, there are nodes with a dominant amount of samples, which also indicates a strong similarity.

Another concern had to do with how fuzzy regions could include as much information as possible from data. We can see from the Table 4.23 that the proposed way of constructing regions and MF results in a much better performance in all experiments, exceeding 90% accuracy threshold as indicated in the Table 4.23. Finally, bootstrap aggregation, used on the 1^{st} step of NF will provide more general clusters that result in corresponding fuzzy regions.

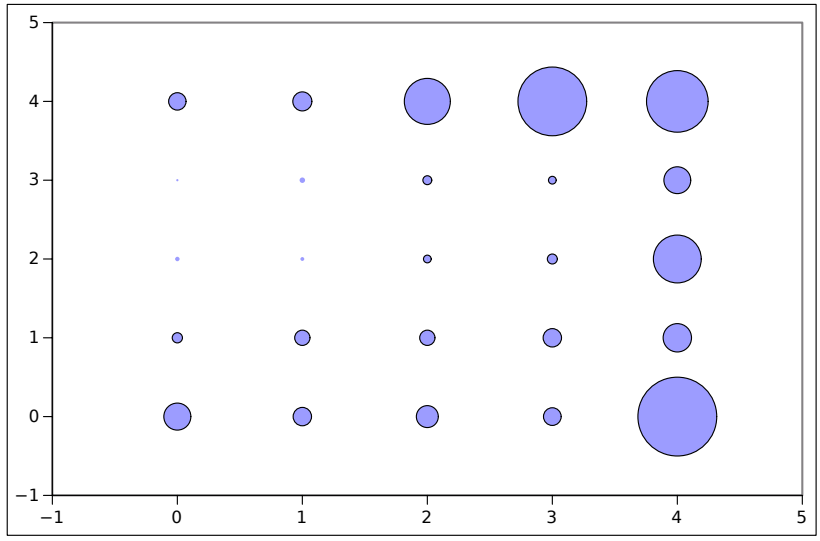
From the experiments, we can see that a lower amount of better generalized fuzzy rules is more acceptable when dealing with the analysis of large-scale datasets. Moreover, the processing time is reduced with better accuracy.

Computational Complexity

The computing speed results are given in the Table 4.24; one can see that the proposed improvements for NF gives an optimal trade-off between accuracy, interpretability, and significant reduction in the number of required computational operations. The 2^{nd} step of NF is the most computationally-intensive one, since each of the extracted fuzzy rules on the 1^{st} step have to be trained against all the samples in the training dataset's defined number of epochs. This also includes linear algebra operations on the matrices that define the membership function for the proposed method. Parallel optimization may provide significant speed up on this step. We also noticed that learning using bootstrap data usually takes 10-20% less



(a) $S_{Vesanto}$ on 1%, 24,595 nodes



(b) $S_{Proposed}$ on 1%, 25 nodes

Figure 4.16: The number of samples in each node during SOM clustering using different optimal size criteria KDDCUP 99.

time than using the full dataset.

Once the model is trained, the rules can be stored in any convenient format. Both simple and Kosko rules require the storage of the center of the fuzzy patch and

Table 4.24: Time in seconds required to learn models and inference new data for dataset without bootstrap

SOM size	Learning, seconds			Inference, 10^{-6} seconds		
	Simple	Kosko	Proposed	Simple	Kosko	Proposed
Proposed, full set	53.64	52.91	469.24	9.44	8.98	40.60
Proposed, bootstrap	57.99	70.66	**379.65**	1.59	1.71	**19.90**
Vesanto, full set	24,141.79	25,301.97	62,998.87	1,510.00	1,550.00	8,660.00
Vesanto, bootstrap	54.33	101.76	1402.59	9.65	11.60	128.00

length of the triangular for each patch, while the proposed method needs to store the center of the fuzzy patch and the inverted covariance matrix for the ellipsoid. We can estimate the amount of memory considering those 39 fuzzy rules that were extracted by our proposed method for 9 features. The estimation is given in the Table 4.25. Ideally, using C++ double format (8 Bytes), the proposed model requires 28 KBytes, which is not a great space. This method does however require some additional overhead to organize storage structure in memory that can add an additional 30-50% to the model size.

Table 4.25: Ideal storage complexity of fuzzy rules for three methods. N_R is a number of rules and N_F is a number of features

Method	*Simple MF*	*Kosko MF*	*Proposed MF*
Needed space	$N_R \cdot 2 \cdot N_F$	$N_R \cdot 2 \cdot N_F$	$N_R \cdot (N_F + N_F^2)$
39-rules model	5,616 Bytes	5,616 Bytes	28,080 Bytes

Performance Comparison with Other Machine Learning Methods

To be able to estimate the utility of the proposed improvements for NF, we compare other Machine Learning methods using the same datasets. Specifically, we used WEKA [149] version 3.6.13 to perform experiments in this subsection. We also selected peer-reviewed and community-accepted Soft Computing and Hard computing methods for the sake of experiments coverage. The full training KDD 99 dataset was used to train the model and further estimate achieved accuracy, time complexity, and the size of the model for HC. The purpose was to see how well it could generalize and describe the data in comparison to the NF approach suggested earlier. Thus, two sets of experiments were performed:

Soft Computing covers a set of methods with inexact solutions such that Bayesian Network, Support Vector Machines (SVM), and Multilayer Perceptron (MLP) besides NF approach. Abraham et al. [41] studied the applicability of SC for network intrusion detection systems, so we think that these methods should be tested

against our method. The results are given in the the Table 4.26. The following classes were used in Weka: *functions.MultilayerPerceptron, functions.libSVM, bayes.BayesNet, bayes.NaiveBayes*.

Table 4.26: Performance of other peer-reviewed Soft Computing methods on KDD 99 dataset

Method	Acc., %	Learn, sec.
MLP (0 layers)	95.1360	418.62
MLP (1 layer)	94.7016	457.80
MLP (5 layers)	95.0051	921.26
MLP (10 layers)	95.4229	1,604.11
SVM	**99.815**	47,394.53
Bayes Network	99.1201	6.10
Naive Bayes	41.2677	1.43

We can see that the results of stand-alone MLP gives lower results ($\approx 95\%$) than the proposed NF method ($\approx 98\%$), which is still considered to be a NN-based method. Moreover, we tested several layers in the MLP and concluded that increased complexity and non-linearity does not necessarily mean increase in classification accuracy. It can be also seen that the proposed NF improvements require more time to train (≈ 379 seconds) to train than simple NF method (≈ 53 seconds). Yet is it still lower than using MLP without hidden layers (≈ 418 seconds). Bayesian methods are quire fast when it comes to learning, yet only Bayesian Network ($\approx 99\%$) can be considered as an accurate one, not Naive Bayes ($\approx 41\%$). Finally, SVM gives the highest accuracy, though the time it took to train this model is not realistic and hardly applicable for any real-life scenario. Moreover, these three models do not provide rule-based models and can not be easily explained.

Hard Computing includes methods that produce crisp models with the help of binary logic and exact values of the attributes rather than inexact approximations. The methods are partially described by Kononenko et al. [232] as the following: linear regression, a set of *rule-based* methods, general decision rules based on specific attributes, and a set of *tree-based* methods that consider decision tree with a set of selected attributes. Results are presented in the Table 4.27. Additionally, the following Weka classes were useg: *functions.SimpleLogistic, rules.JRip, rules.NNgre, trees.RandomForest, trees.RandomTree, trees.J48*. Extracted rules are numerical and do not involve any kind of abstraction like linguistic rules. So, either a large amount of rules are generated that hard to prune or a lower number of trees that require computation-intensive pruning or stochastic evaluation.
Note: k-NN did not succeed in inferring the data from the build model, probably

Table 4.27: Performance of peer-reviewed Hard Computing Computing methods on KDD 99 dataset

Method	Acc., %	Learn, sec.	Model size
Linear Reg.	95.0569	1904.88	$2 \cdot 9$ parameters
RIPPER	99.8589	488.25	18 rules
k-NN based	n/a	1742.26	17,453 exemplars
Random Tree	99.9324	7.67	631 tree size
Random Forest	99.9290	82.42	10 trees
C4.5	**99.9126**	24.74	157 trees/79 leaves

due to the overwhelming amount of generated exemplars and enormous amount of required memory. At this point, we can say that this is a weakness of many conventional ML methods developed a decade ago, since smaller sets were initially tested with thousands or tens of thousands of data.

Summary: We have proposed a way that NF can be improved in order to facilitate a construction of firewall rules using fuzzy logic based on the large-scale analysis of network traffic. We made a synergy of optimal SOM size determination, the most appropriate method of fuzzy regions construction, and the corresponding MF that can be derived to incorporate all possible information on the 2^{nd} step of NF. The new method showed a great improvement in performance and required less learning and inference time in comparison to classical approaches. Additionally, we studied that bootstrapping on the 1^{st} stage of NF may result in a lower amount of rules and more generalized classification model based on the fuzzy rules, which is important to the experiment. The achieved accuracy of the proposed improvement was 98.787% when using 39 rules against 98.790% with 10,231 rules when applying original method. Therefore, it can be used as a part of decision support system and firewalls to strengthen the security of networks. Computational time and space complexity make it possible to use on embedded hardware solutions. In future research, we will investigate the utility of non-parametric models for elliptic regions estimation and the possibility of online retraining of the fuzzy layer without complete re-learning of the whole NF model. Also, by parallel processing a model, training can be optimized using CPUs with a higher number of computational cores, especially on the intensive 2^{nd} step of Neuro-Fuzzy.

4.4 Web Application Firewalls[5]

We focus here on aspects of building multinomial classification Neuro-Fuzzy (NF) models for the detection of attacks on web applications using different properties

[5]The main ideas of this chapter are published under the contributions [372, 377]

of HTTP requests. Mamdani-type NF is a model which is specifically designed for classification problems, where each fuzzy rule denotes a specific group of samples that can be denoted with a fixed label as studied by Kosko and Chen [233, 95].

4.4.1 Datasets

Our main motivation was to perform fast and accurate classification of web attacks using a single NF model. For this purpose, we used the ECML/PKDD 2007 Web Attacks Discovery Challenge [25]. However, it is important to benchmark multinomial classification on data with different properties to be able to understand the limitations of the proposed and other methods. We acquired several datasets as listed below from UCI Machine Learning Repository [16]. The data were subject to thorough criteria such that absence of documented mistaken, missing data, usage in literature before, etc. Properties of the mentioned above datasets are given in the Table 4.28.

1. **Isolet Data Set** represent data collected from different speakers that represent the letters A-Z (classes) of the English alphabet. These features include spectral coefficients, sonorant features, etc. The dataset was originally published by UCI in 1994.

2. **Wine Quality Data Set** represents data that describe results of the physiochemical quality assessment of red and white vinho verde wines from Portugal. Features include different measurements such that pH level, density, etc. The dataset was published by UCI in 2009.

3. **PKDD 2007 - Web Traffic Data Set** contains raw XML dumps of HTTP requests that characterize different types of web-attacks, as well as normal valid requests as described by Gallagher et al. [157]. Each sample includes the parameters of the server, OS, MySQL, etc. To cover different types of attacks, we manually analysed indicators of these attacks and extracted the corresponding features, since the dataset consisted only of raw data. The following features were extracted: *os, webserver, runningLdap, runningSqlDb, runningXpath, method, protocol, strlenUri, strlenQuary, strlenHeAder, strlenHost, strlenAccept, strlenAcceptCharset, strlenAcceptEncoding, strlenAcceptLanguage, strlenReferer, strlenUserAgent, strlenUACPU, strlenVia, strlenWarning, strlenCache-Control, strlenClient-ip, strlenCookie, strlenFrom, strlenMax-Forwards, strlenConnection, strlenContent-Type, entropyUri, entropyQuery, countExe, countShell, countSelect, countUpdate, countWhere, countFrom, countUser, countPassword, countOR, countPs, countGcc, countXeQ, countDir, countLs, countQueryArgs*. This was inspired by Pachopoulos et al. [306], who managed to extract 216 features with nearly

the same accuracy of 77%. The dataset has been criticized before because it is artificial and contains a very skewed class distribution, so we additionally used re-sampling to get a more reliable distribution. Abdi et al. [37] indicated the importance of having balanced classes when training machine learning methods. In most datasets related to attacks detection, the normal traffic is the majority, while attacks are sometimes are underrepresented. Therefore, we used the filter *Resample* in *Weka* that produces a random subsample of data using the parameters of the desired class distribution.

4. **Pen-Based Recognition of Handwritten Digits Data Set** is a set of digits (250 samples) that were collected using pressure sensitive tablet from 30 writers. The dataset was published by UCI in 1998.

5. **Connectionist Bench (Vowel Recognition - Deterding Data) Data Set** represents a set of steady state vowels of British English. The dataset was created in 1989.

Table 4.28: Properties of the dataset. N_S is a number of samples in a set, N_F is a number of features, N_C is a number of classes, e_0 and e_1 represents the 1^{st} and 2^{nd} biggest eigenvalues.

Dataset	N_S	N_F	N_C	e_0	e_1
Wine-red	1,599	11	10	3.0260	1.9138
Wine-white	4,898	11	10	3.1624	1.5752
Isolet1+2+3+4	6,238	617	26	119.1251	54.8089
PKDD 2007	50,000	43	8	4.3303	2.8711
Pendigits	7494	16	10	4.6780	3.1912
Vowels	528	10	11	2.3316	2.1420

4.4.2 Experimental Design

Each experiment included: SOM grouping, extracting three types of fuzzy regions parameters (rectangular, Kosko, proposed method), and models training. To estimate the speed of the execution, the relative times of the experiments were measured for all samples in the training set.

4.4.3 Performance Evaluation

Both binary and multinomial classification methods require different approaches to compare their performance:

- *For binary classification methods* a number of metrics were designed, such as *Precision, Recall (Sensitivity)* and *Specificity*. Those measure the amount of correctly classified instances with respect to one or another class.

- *For multinomial classification methods* metrics have to be calculated for each of the classes using some kind of averaging, which is out of our scope since the classes can be considered relatively balanced.

Sokolova et al. [397] performed a systematic analysis of performance measures for different types of classification tasks. We can see that *Accuracy* evaluates overall effectiveness in both methods, so it will be considered later.

The experiments were designed to show the performance of the proposed scheme. Two types of performance metrics were utilized for our method. (1) *Regression-based* accuracy using the defuzzifier value: Mean Absolute Error (MAE), Relative Absolute Error (RAE), and Mean Absolute Percent Error (MAPE).

$$MAE = \frac{1}{N_S}\sum_{i=1}^{N_S}|y_i - d_i|, \tag{4.5}$$

$$MAPE = \frac{1}{N_S}\sum_{i=1}^{N_S}|\frac{y_i - d_i}{d_i}| \cdot 100\%, \tag{4.6}$$

$$RAE = \frac{\sum_{i=1}^{N_S}|y_i - d_i|}{\sum_{i=1}^{N_S}|d_i - \bar{d}|} \tag{4.7}$$

$$Acc = \frac{N_P}{N_S} \tag{4.8}$$

where y_i - the output of the NF defuzzifier for a particular data sample, d_i - the actual class of the sample, w_{ij} - weight of a particular rule, and μ_{ji} - MF value of a particular rule, N_S - number of given data samples, and N_P - number of properly classified samples according to $max-min$ inference principle [233] in Mamdani-type rules. (2) *Classification-based* which estimates the *Accuracy*, or how well the rules selected by MAX-MIN principle classifies the data samples by calculating the percentage of correctly classified samples. We refer to the second measure when talking about classification accuracy. This type of accuracy was also used in other experiments using the WEKA tool.

4.4.4 Results & Analysis

The results below are divided into two blocks. First, we evaluated the newly proposed method against the datasets. Second, we compared the achieved accuracy against community-accepted ML methods.

Classification Accuracy

The performance results of the proposed improvements are given in the Table 4.29. Some of the datasets contain training and testing sets, yet some only testing. Thus, we concentrate here only on training samples, and try to build a model that describes the data by means of fuzzy rules. Additionally, using the improvements suggested earlier for binary classification problems [373, 374], we are able to tune the accuracy of multinomial classification significantly.

Table 4.29: Performance comparison of NF with a single linear output combiner

Fuzzy patches	**MAE**	**RAE**	**MAPE**	**Acc**,%	**MAE**	**RAE**	**MAPE**,%	**Acc** ,%
	Dataset: Wine-red				**Dataset: Wine-white**			
Rect. [233]	0.656	0.961	11.505	41.338	0.749	1.117	13.796	37.260
Kosko [233, 120]	0.694	1.016	12.346	43.089	0.692	1.031	11.859	27.623
Proposed [373, 374]	0.566	0.828	9.582	**68.667**	0.610	0.916	11.076	**59.064**
	46 rules				**71** rules			
	Dataset: Isolet1+2+3+4				**Dataset: PKDD 2007**			
Rect. [233]	6.620	1.018	135.858	2.837	3.827	1.907	108.343	9.300
Kosko [233, 120]	6.467	0.995	131.042	20.984	2.094	1.043	99.889	12.131
Proposed [373, 374]	5.105	0.785	37.159	**64.363**	1.440	0.717	66.319	**52.570**
	174 rules				**48** rules			
	Dataset: Pendigits				**Dataset: Vowels**			
Rect. [233]	2.203	0.873	92.700	23.031	2.572	0.943	97.969	26.515
Kosko [233, 120]	3.298	1.307	72.639	79.837	2.629	0.964	89.949	57.575
Proposed [373, 374]	0.116	0.046	5.358	**97.811**	2.629	0.272	22.046	**75.757**
	102 rules				**53** rules			

It can be seen that our method works well on all datasets and shows consistently high accuracy in comparison to simple rectangular patches and the Kosko method. Moreover, a method with simple rectangular patches produces a high error rate, which indicates the goodness of fit of the proposed improvements. On the *Isolet* dataset our method achieved $\approx 64\%$, while rectangular fuzzy patches give an almost random result of $\approx 3\%$. The last approximation means that the classifier failed to classify properly nearly every instance in the dataset.

Comparison to Other Machine Learning Methods

To be able to independently evaluate the accuracy of the proposed NF improvements, we decided to perform several tests using the implementation of ML methods available in WEKA [149] version 3.6.13. All the chosen methods were mentioned in the Section 2 and are peer-reviewed by the ML community. The following experiments are performed: (i) binary classification methods for multinomial problems, (ii) multinomial classification methods, and (iii) influence of

non-linearity in NN as compared to single-layer NF.

Performance on binary classification methods was estimated using the following WEKA functions: *J48, libSVM, SimpleLogistic, MultilayerPerceptron*. MLP had 1 hidden layer and 500 training epochs. The results are given in the Table 4.30.

Table 4.30: Accuracy of binary classifiers in Weka, %

Dataset	Decision Tree	SVM	Logistic Reg.	$\text{MLP}_{1\ l}$
Wine-red	**60.537**	58.036	59.787	58.536
Wine-white	**58.677**	56.227	53.246	51.776
Isolet1+2+3+4	82.654	95.014	**95.832**	7.630
PKDD 2007	83.925	**91.491**	48.960	21.049
Pendigits	95.823	13.250	**96.530**	38.497
Vowels	78.787	**88.068**	69.128	29.545

We can say that there is no consistently highest accuracy method for all 6 studied datasets. However, *MLP* with 1 hidden layer gives much worse results than corresponding NF with 1 combiner layer. Furthermore, *SVM* failed to classify a majority of the samples in *Pendigits* dataset. *Decision Tree* method J48 performed relatively well on a majority of the datasets, yet the size of the trained model is able to reach multiple hundreds and thousands of leaves, which may break the generalization. *Logistic Regression* performs well with an accuracy nearly equal to *SVM*; in the case of the *Isolet* dataset with 617 features however, it takes too much time to estimate the parameters for each class regression model.

Performance on multinomial classification methods was estimated following the WEKA functions: *IBk, RandomForest, NaiveBayesMultinomial, BayesNet*. IBk was used with a number of neighbours $k = 1$. The results are presented in the Table 4.31.

Table 4.31: Accuracy of multinomial classifiers in Weka, %

Dataset	k-NN	Random For.	Naive Bayes	BayesNet
Wine-red	63.727	**69.731**	43.902	58.286
Wine-white	63.842	**68.395**	39.424	47.856
Isolet1+2+3+4	88.233	86.325	84.081	**90.093**
PKDD 2007	87.263	**89.861**	36.373	51.995
Pendigits	**99.292**	99.146	83.480	88.604
Vowels	**99.053**	95.075	66.287	61.553

All multinomial classification methods produce consistently good accuracy on all datasets. Exceptions are *Naive Bayes* and *Bayes Network* that are able to overcome NF with the proposed improvements only on the *Isolet* dataset. Moreover, *Naive Bayes* produced half of the results with an accuracy lower than 50%, so we can conclude that this method does not generalize well. Finally, our observation is that the execution of the binary classifiers takes much more time to train that any of the originally designed multinomial classifiers.

Influence of non-linearity in NN was estimated using various numbers of hidden layers (0-10) in a single MLP with multiple outputs. However, it makes an impact on the overall complexity of the model. Since NF is a Neural Network-based architecture, it is important to know whether increasing the non-linearity actually improves in classification accuracy or not. The results of these experiments are given in the Table 4.32.

Table 4.32: Accuracy of MLP with respect to non-linearity in Weka (100 epochs), %

Dataset	**MLP**$_{0\ l}$*	**MLP**$_{1\ l}$	**MLP**$_{5\ l}$	**MLP**$_{10\ l}$
Wine-red	58.098	58.786	58.536	**58.787**
Wine-white	50.592	52.021	53.348	**54.532**
Isolet1+2+3+4	**95.142**	7.566	24.687	88.393
PKDD 2007	46.326	20.328	46.605	**50.608**
Pendigits	93.047	38.417	91.419	**94.448**
Vowels	55.303	23.295	64.962	**77.651**

**MLP with 0 hidden layers means that there are N_C separate linear combiners of input features set into output neurons denoting each class. Basically, we have multiple independent Neural Networks.*

We can see that a set of multiple MLPs generally performs better than a single MLP with multiple hidden layers. Yet this creates a large computing overhead needed to train N_C separate models. Furthermore, 1-layer MLP produces accuracy higher than 50% only on both *Wine* datasets, which is an extremely poor performance. These results are also consistent with the original research performed on the datasets by Cortex et al. [105]. Finally, we can see that 10-layer MLP performs nearly as well as 1-layer NF with the proposed improvements, except in *Isolet* dataset. Finally, we can say that our method shows consistently good accuracy as compared to multiple other methods, while also being able to learn a model quickly. The results for the *Vowels* dataset are similar to the performance achieved by Thimm et al. [414] when considering this dataset as a binary classification problem with outputs equal to -1 and +1.

Finally, we can see that the results obtained by the NF methods with the proposed improvements are consistent with other methods. The huge number of features in ISOLET dataset makes it impossible to classify data with a higher degree of accuracy. Apparently, the challenge is that the data are very complex and have non-linear relations for different classes. Moreover, for the PKDD 2007 dataset, MLP with 10 hidden layers shows the same accuracy as our method to a much lower degree of non-linearity. On the WINE dataset, our method performs even better than C4.5.

Note. We noticed that MLP implementation in Weka uses *"one-hot"* encoding that creates enormous overhead with the number of hidden layers > 1 since multiple outputs need to be evaluated. As a result, skewed classes distribution in the *One-against-All* training results in significant errors.

Summary: This contribution proposed improvements towards the application of NF in multinomial classification problems, such as web attacks detection. The majority of Network Forensics applications requires one to find not only *"benign"* or *"malicious"* activity patterns, yet also to distinguish between multiple *"malicious"* patterns. However, a conventional single-output Neural Network-based method works with either two sets of classes or alternatively requires additional outputs per class. In order to overcome multiple outputs encoding in the NF, we proposed bounding the clustering results of SOM for better statistically-sound fuzzy rules parameters as well as apply a modified Gaussian membership function. Then, we suggested using a single-output mode for reduced overhead and training time. The Corresponding Center of Gravity defuzzifier was modified to be compliant with Mandani-type rules as well as to incorporate class labels. The results were tested on a range of various datasets with completely different properties, and the accuracy is only comparable to a performance achieved by 10 layers multi-output MLP implementation in Weka. Additionally, we studied the performance of binary and multinomial classification methods and can say that multinomial classification models generally perform better and faster on all sets than ensemble binary classifiers. Not all of them can produce understandable models however, as in case with NF. Thus, we believe that it is better to use multinomial NF with proposed improvements for web attacks differentiation to achieve consistent classification results by fuzzy rules.

4.5 Network Forensics Readiness[6]

Soft Computing (SC) methods are known to be able to extract both accurate and interpretable classification models from Network Forensics data, Neuro-Fuzzy (NF)

[6]The main ideas of this section are published under the contributions [376, 432, 433]

in particular, as studied by Anaya et al. [59]. This capability is important when dealing with digital forensics investigations and incident responses within that context. In 2014, Al-Mahrouqi et al. [49] presented a Network Forensics readiness and security awareness framework. We can see based on that research that it is not only important to preserve data properly, but also to analyze it meaningfully and extract relevant knowledge. Adeyemi et al. [44] studied features that are relevant for network forensics investigations with respect to different stakeholders. One can see that there are many comprehensive characteristics that can be helpful in differentiating between normal traffic and attacks.

4.5.1 Datasets

To perform a comprehensive evaluation and assessment of the proposed methods, several datasets were used: HIGGS (1), SUSY (2), Record Linkage Comparison Patterns (3), KDDCUP 1999 10% and KDDCUP 1999 (4) [250]. They all are publicly available and can be accessed using the UCI Machine Learning Repository [16].

Add. 1: **HIGGS Data Set** was published by Baldi et al. [71] in 2014. It presents a classification problem in distinguishing between the signals from Higgs-Boson particles and background radiation. It consists of the 28 real attributes values of different metrics related to signal processing. In our experiment, these datasets were adopted for training without major modifications of the contents. There are 5,170,877 samples of the 1_{st} class and 5,829,123 samples of the $2nd$ class out of 11,000,000 samples in total.

Add. 2: **SUSY Data Set** was published by Baldi et al. [71] in 2014 and presents the classification problem in distinguishing between the signal value produced by particles and background noise. All 18 attributes are real values. This dataset was not modified when used in our experiments. There are 2,712,173 samples of the 1^{st} class and 2,287,287 samples of the 2^{nd} class out of 5,000,000 samples in total.

Add. 3: **Record Linkage Comparison Patterns** dataset was published by Schmidtmann et al. [359] in 2009. It covers a classification problem of identifying whether separate records belong to the same person. This dataset was modified during the pre-processing. In particular, the attributes *cmp_fname_c2* and *cmp_lname_c2* are missing in 98.19% and 99.95% of the records respectively. These two attributes were removed from the dataset. Moreover, $< 0.2\%$ of the data records include other missing features, so the records with missing attributes were removed as well. The resulting preprocessed dataset consists of 5,734,488 records (5749132 initially) and 9 real-valued features (11

initially). Overall, there are 20,887 samples of the 1^{st} class and 5,713,601 samples of the 2^{nd} class.

Add. 4: **KDD CUP 1999 10%** and KDD CUP 1999 FULL are well-known datasets that were released by Kibler et al. [107] in 2000 as a part of KDD Cup 1999 challenge "Computer network intrusion detection". They represent a set of 41 characteristics of the network packets characterized as "bad" or "good" connections. This is the task of Intrusion Detection, and the model derived from the data can be used to prevent any malicious activity. The features values are real, symbolic, binary, and integers. These datasets were modified during the pre-processing step. In particular, the Feature Selection was done based on the method proposed by Nguyen et al. [294] in 2010. It implies that we selected only features that contribute to the classification of benign and malicious connections [id - name]: 5 - *src_bytes*, 6 - *dst_bytes*, 10 - *hot*, 12 - *logged_in*, 14 - *root_shell*, 22 - *is_guest_login*, 29 - *same_srv_rate*, 37 - *dst_host_srv_diff_host_rate*, 41 - *dst_host_srv_rerror_rate*. According to the authors, these features provide classification without much loss of accuracy in comparison to the original dataset. The KDD Cup 1999 10% set includes 97,278 samples of the 1^{st} class ("normal") and 396,743 samples of the 2^{nd} class ("attacks") out of 494,022 samples in total. The full KDD Cup 1999 set consists of 972,781 samples of the 1^{st} class ("normal") and 3,925,650 samples of the ^{2}nd class ("attacks") out of 4,898,431 samples in total.

The properties of these datasets are presented in Table 4.33. The preprocessing was first applied to get suitable data for learning from. The number of suggested SOM nodes are included as well for both the Vesanto method and our proposed one.

Though each cluster can have samples from both classes, the actual number of clusters will be in between the S and $nC \cdot S$, where S is a number of suggested SOM nodes. The proposed scheme for the optimal amount of SOM size reduces the total number of clusters extracted by SOM.

4.5.2 Experimental Design

To apply the proposed improvements of the Neuro-Fuzzy method, and to verify our hypothesis mentioned in Section 1, the following experiments were performed: (1) the estimation of an optimal SOM size based on correlation and eigendecomposition, and analysis of the overfitting / underfitting and spread of fuzzy rules according to SOM clustering results; (2) training of the NF method using a full dataset, estimating the performance of the trained dataset; (3) generation of a Bootstrap

Table 4.33: The properties of the datasets used in the experiments are based on the data obtained from the statistical program PSPP. The columns are: N_S - number of data samples in the dataset, N_F - number of features, $E0$ and $E1$ - the 1^{st} and the 2^{nd} biggest eigenvalues of the dataset, $\bar{r}$ - average Pearson Correlation Coefficient, S_P - proposed optimal size of the SOM grid, and S_V - an optimal size of SOM, according to Vesanto, S_{Vlower} - the lower boundary of the Vesanto method, and S_{Vupper} - the upper boundary of Vesanto method.

Dataset	N_S	N_F	$E0$	$E1$	$\lvert\bar{r}\rvert$	S_P	S_V	S_{Vlower}	S_{Vupper}
HIGGS	11,000,000	28	4.164	1.864	0.066	11	37,040	9,260	148,160
SUSY	5,000,000	18	4.806	3.772	0.181	14	14,246	3,562	56,984
RL	5,734,488	9	1.937	1.501	0.114	10	15,450	3,863	61,800
KDD	4,898,431	9	1.819	1.433	0.057	7	14,054	3,514	56,216
KDD_10%	494,022	9	6.998	1.000	0.583	25	24,595	6,149	98,382

Aggregation subsample for performance estimation; (4) comparison of different performance metrics for experiments 2 and 3 in order to prove or deny the hypothesis; (5) comparison to other scientifically-proven ML classification methods, such as SVM, Bayesian Network, C4.5, Random Tree, and Multilayer Perceptron. In our experiments, we used the two stages of NF architecture with the improvements proposed earlier: (i) SOM is initialized with dimension parameters and used to group data samples according to their classes; (ii) parameters of fuzzy clusters are extracted forming fuzzy rules, while single-layer ANN is used to tune the corresponding weights of each fuzzy rule. The initial learning rate for SOM (it is decreasing over time) and for ANN NF was chosen to be equal to 0.1.

Virtual Dedicated Server was used to perform all the experiments. Practical details about computing platforms are given in the A.2. To estimate the relative speed of execution, time requirements of the experiments include: (1) estimation of an optimal SOM size, (2) training of the three models using full datasets, and (3) the estimation of five accuracy metrics mentioned earlier for all the samples in the training set. The approximate time frame required for each of the experiments are presented in Table 4.34 for the proposed improvements.

Table 4.34: Amount of time in *minutes* required to perform a complete experiment on each dataset for the proposed improvements

HIGGS	**SUSY**	**RL**	**KDD Cup 1999**	**KDD Cup 1999 10%**
185	99	72	55	15

We also noticed that learning using bootstrap aggregation usually takes 10-20% less time than from the full dataset. To compare, we utilized the Vesanto [423]

method on KDD 10% dataset. The required time was increased dramatically (in *minutes*): 402 for the rectangular patches with triangular MF, 421 for the Kosko elliptic patches with triangular MF and 1,049 for the proposed method of fuzzy patches construction and MF derivation. Consequently, we can conclude that the original methods (NF by Kosko and SOM size determination by Vesanto) are hardly applicable for large-scale datasets and take an enormous amount of resources to learn the fuzzy model.

4.5.3 Performance Evaluation

To estimate the accuracy of the models on the datasets mentioned earlier, we decided to use several error measures due to the fact that the sample belongs to a particular class and had been foretasted by the models. The following two groups of metrics are used: the first is the regression-based accuracy of the model Mean Absolute Error (MAE), Relative Absolute Error (RAE), and Mean Absolute Percent Error (MAPE) based on the value of the center of gravity defuzzifier by Kosko [233]:

$$y_i = \frac{\sum_{j=1}^{N_R} d_i \cdot \mu_{ji} \cdot w_{ji}}{\sum_{j=1}^{N_R} \mu_{ij} \cdot w_{ij}} \tag{4.9}$$

where N_R is a number of extracted rules, y_i - the output of the NF, d_i - the actual class of the sample, w_{ij} - the weight of a particular rule, and μ_{ji} - the MF value of a particular rule.

The second one estimates how well the rules selected by max-min principle classify the data samples by calculating the percentage of correctly classified samples Acc.

$$MAE = \frac{1}{N_S} \sum_{i=1}^{N_S} |y_i - d_i| \tag{4.10}$$

$$MAPE = \frac{1}{N_S} \sum_{i=1}^{N_S} |\frac{y_i - d_i}{d_i}| \cdot 100\% \tag{4.11}$$

$$RAE = \frac{\sum_{i=1}^{N_S} |y_i - d_i|}{\sum_{i=1}^{N_S} |d_i - \bar{d}|} \tag{4.12}$$

$$Acc = \frac{nP}{N_S} \tag{4.13}$$

where N - number of given data samples and nP - number of properly classified samples according to $max - min$ inference principle, according to Kosko [233] in Mamdani-type rules. Naturally, we refer to the second measure when discussing

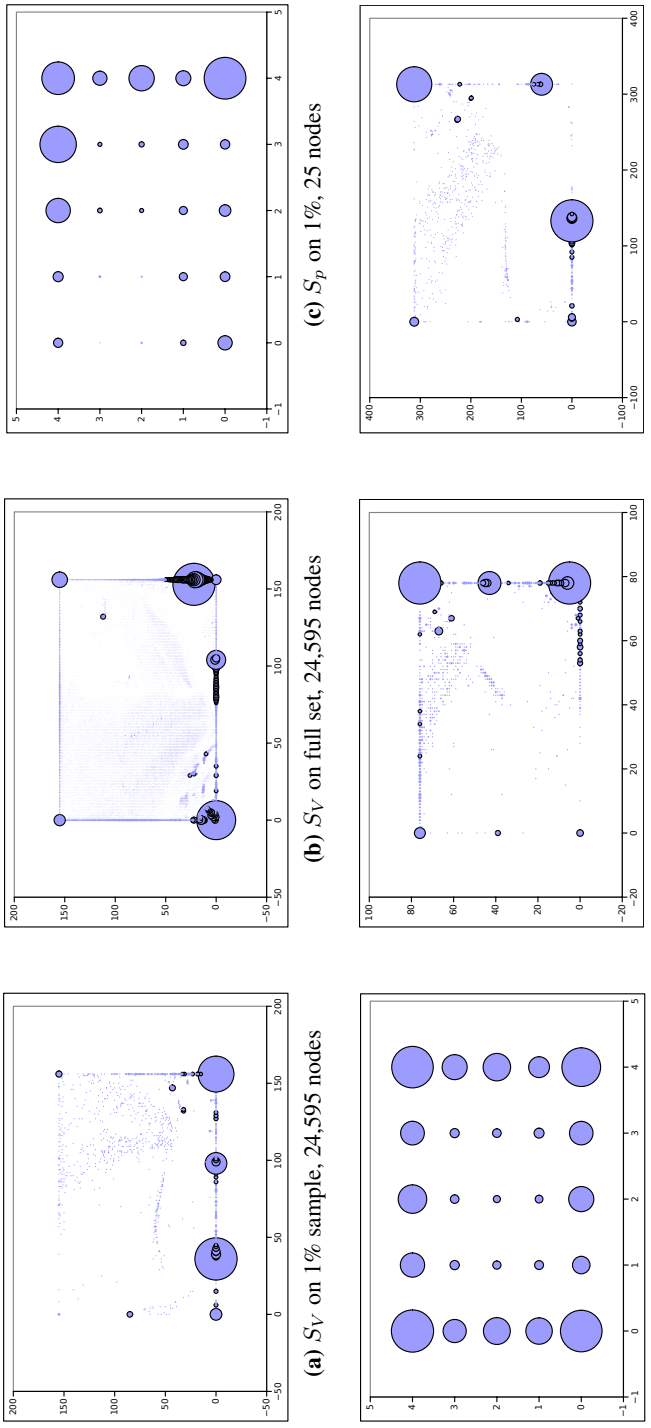
classification accuracy. To support the applicability of the proposed method, one can refer to the regression metrics as well.

4.5.4 Results & Analysis

In this Section, we present the results of the proposed method on the dataset to verify suggested improvements and confirm the hypothesis. Though we performed multiple experiments, as described above, we chose to represent them in fewer tables for analysis convenience. Also, we did a preliminary feasibility study earlier on KDD 10% dataset [375]; the results were promising. Each table with performance metrics includes bootstrap aggregation results as well as training on the full dataset.

Optimal Self Organizing Map Size

The first stage of NF is defined by the quality and complexity of SOM clustering. To show the difference between our method and the Vesanto method, we used bubble plots in Figure 4.17 to represent the distribution and amount of samples clustered in the SOM grid. The size of the biggest clusters are limited by the size of the bubble; the purpose of this plot is to show a tentative distribution. It can be clearly seen that in the case of the Vesanto method, the number of empty nodes is enormous, while in the proposed method there are only few nodes with 0 data samples. Additionally, it takes much more time to perform clustering using Vesanto at an optimal size since the number of nodes to be updated is several hundred times more numerous than in the proposed one. Finally, the Vesanto model will not result in a human-understandable fuzzy classification model due to the great number of extracted rules. Consequently, we need to look at each rule's merit definition in the Vesanto Method in order to reduce the number of rules to a reasonable one, according to subjective quality.



(a) S_V on 1% sample, 24,595 nodes

(b) S_V on full set, 24,595 nodes

(c) S_p on 1%, 25 nodes

(d) S_p on full set , 25 nodes

(e) S_{Vlower} on 1% sample, 6,149 nodes

(f) S_{Vupper} on 1% sample, 98,382 nodes

Figure 4.17: The number of samples in each node during SOM clustering using different optimal size criteria KDDCUP 99 10% set.

Clearly, SOM size based on the Vesanto method is large, and there are several dominating clusters with a majority of the samples located within them. Other nodes contain a negligible amount of data samples and do not provide much merit in the final classification model. As a result, the accuracy degrades and the model becomes highly overfitted compared to the suggested model.

Classification Accuracy Using Proposed Self Organizing Map Size

To reduce complexity and improve generalization in the SOM grouping phase, Bootstrap Aggregation was used on 1% of randomly selected samples from the dataset. The following experiments were used to evaluate their performance: (i) for the bootstrap method, we performed 5 runs of SOM training on 1% of sets to determine the best sequence for the whole of NF training. What this means is that 5 datasets in total were generated from the original dataset. The performance was then estimated using the model trained from the best sample and using the whole training dataset. (ii) For experiments without bootstrap, and using the proposed SOM size determination method, we used 5 runs to determine the best accuracy of SOM training on whole dataset. (iii) For the Vesanto method, we used a single SOM training run since the experiment time was too enormous to perform more trials. The results shown in the Table 4.29 include the following abbreviations: M denotes used method: S - simple rectangular patches with triangular MF, K - elliptic patches with Kosko MF, P - proposed construction of the patches with the corresponding MF. The example of the fuzzy rule's parameters and corresponding visualization is given in the Appendix.

Table 4.35 shows that accuracy on the HIGGS dataset reaches 67.961% using the full dataset and 61.150% using bootstrap aggregation. To compare, we can say that according to Kheirkhahan et al. [222] the AUC on 1% of this dataset using 5-hidden layer perceptron with 300 nodes is 72.41%, which can be treated as decent for that complex model. Furthermore, according to Whiteson et al. [437] the network with two hidden layers can increase accuracy from about 69.90% to nearly 70.6%, which increases the computation time dramatically on the whole dataset. Also, the accuracy on the SUSY dataset was 67.961% using bootstrap aggregation on the proposed method. In comparison, according to the study by Triguero et al. [417], the accuracy of K-NN win $k = 1$ was 68.99%. Moreover, according to this study, the highest accuracy using k-NN was 99.60% against 99.758% achieved by the proposed method on the Record Linkage dataset. This is on the edge, since there is very little difference from 100%. Also, we can state that the data most likely forms multivariate distribution that fits an ellipse quite well, since the accuracy on the rectangular patches is extremely poor.

Table 4.35: Performance comparison (regression, classification) of the proposed method with and without bootstrap aggregation on the dataset.

M	Full dataset				Boostrap aggregation			
	MAE	**RAE**	**MAPE**,%	**Acc**,%	**MAE**	**RAE**	**MAPE**,%	**Acc** ,%
Dataset: HIGGS								
S	0.374	0.754	23.719	63.963	1.511	3.033	98.949	46.660
K	0.457	0.921	22.878	65.139	0.500	1.003	36.676	49.027
P	0.398	0.802	37.829	**67.961**	0.422	0.847	27.136	**61.150**
	16 rules				**16** rules			
Dataset: SUSY								
S	0.400	0.807	20.276	62.364	0.374	0.754	23.719	63.963
K	0.500	1.000	40.756	61.686	0.457	0.921	22.878	65.139
P	0.393	0.792	37.298	**67.669**	0.398	0.802	37.829	**67.961**
	24 rules				**24** rules			
Dataset: Record Linkage								
S	1.028	141.739	51.534	1.460	0.996	137.276	49.818	0.364
K	0.003	0.470	0.327	99.743	0.001	0.101	0.072	**99.977**
P	0.005	0.755	0.276	**99.752**	0.005	0.714	0.262	99.758
	16 rules				**16** rules			
Dataset: KDD Cup 1999 full set								
S	1.800	5.657	99.946	18.251	1.790	5.623	98.883	19.137
K	0.432	1.360	27.933	80.211	0.491	1.544	28.467	72.511
P	0.147	0.463	9.434	**94.877**	0.109	0.344	5.809	**88.215**
	12 rules				**12** rules			
Dataset: KDD Cup 1999 10% set								
S	1.124	3.556	66.020	14.059	0.198	0.627	19.425	73.135
K	0.454	1.437	26.914	91.710	0.205	0.650	10.896	90.455
P	0.077	0.244	6.029	**94.571**	0.118	0.374	5.970	**98.787**
	50 rules				**39** rules			

Table 4.36: Performance comparison (regression, classification) of the Vesanto method on the KDD CUP 1999 full dataset without bootstrap aggregation.

Method	**MAE**	**RAE**	**MAPE**,%	**Acc**,%	**Train. Time**, sec
Dataset: **SUSY**					
Simple	0.4318	0.8699	32.1105	66.5670	345,487
Kosko	0.4960	0.9992	38.2749	62.4708	468,894
Proposed	0.3412	0.6874	29.5479	**68.5518**	2,350,348
29,166 extracted rules					
Dataset: **Record Linkage**					
Simple	0.0868	11.9575	4.4774	82.7559	268,642
Kosko	0.0037	0.5037	0.3358	**99.9821**	310,489
Proposed	0.0012	0.1619	0.0643	99.8022	1,010,158
16,843 extracted rules					
Dataset: **KDD CUP 1999 full set**					
Simple	0.1312	0.4121	12.3240	82.9980	127,745
Kosko	0.8053	2.5300	40.4663	80.5873	344,220
Proposed	0.5788	1.8184	29.3671	**98.9855**	756,837
15,081 extracted rules					
Dataset: **KDD CUP 1999 10%**					
Simple	0.1684	0.5323	14.7287	38.2728	24,141
Kosko	0.8171	2.5836	41.6208	80.0831	25,301
Proposed	0.5797	1.8329	29.1953	**98.7909**	62,998
10,231 extracted rules					

Classification Accuracy Using Vesanto Self Organizing Map Size

To compare with the proposed enhancements, we also performed the experiments using the Vesanto metric from Equation 2.13. The results are presented in Table 4.36. It can be seen that the results for the KDD Cup 1999 10% set using the proposed method from Table 4.35 are the same as when uing bootstrap aggregation; resulting in 39 rules, against 10,231 from the Vesanto method. It is clear that the complexity of the model based on Vesanto is much higher, yet the classification results are the same. The results achieved here are slightly better than using bootstrap and the proposed SOM size, yet the complexity is too high and inapplicable for real life scenarios.

Influence of Self Organizing Map Training on Fuzzy Patches Allocation

To understand the influence of bootstrap aggregation on SOM clustering and on the second stage of NF, we visualised the way fuzzy patches are located. Since it is not possible to visualize all 9 features used for the *KDD* dataset, we took only the two features that contribute most to classification, according to *ReliefF* and *Information Gain* quality measures. These are the 1^{st} and 2^{nd} attributes, which are called "*src_bytes*" and "*dst_bytes*" respectively. The results of the visualization (scatter plot)

of the fuzzy rules center using only these two attributes are given in Figure 4.18. The color of the points indicates class label, while the shape represents the SOM learning method: the full set and bootstrap aggregation.

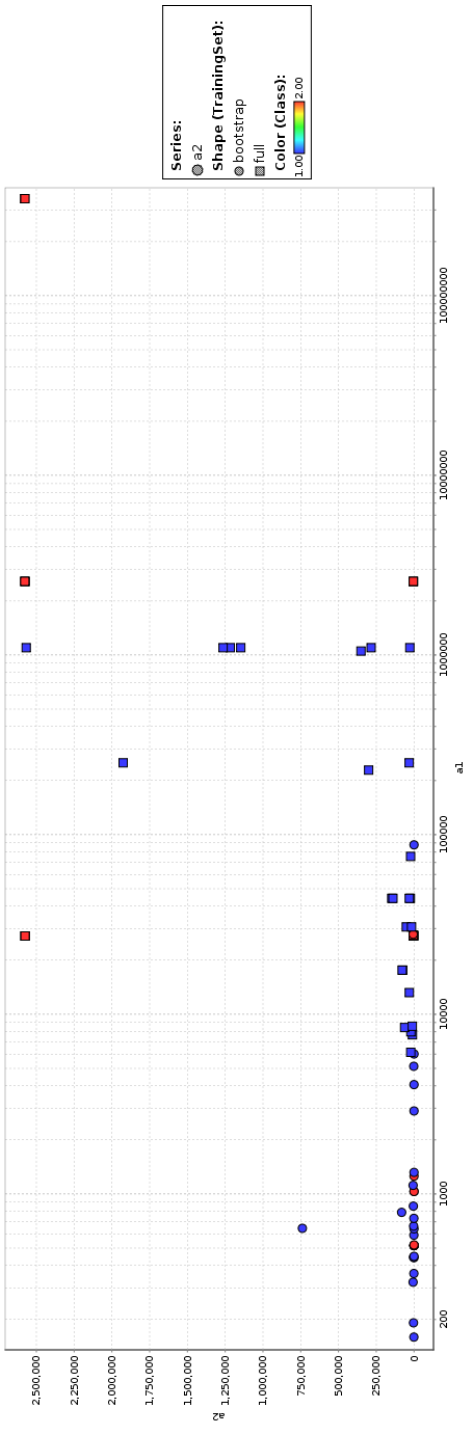


Figure 4.18: Allocation of the centres of fuzzy rules for the KDD 10% dataset for both classes for training with the full dataset and bootstrap set respectively using RapidMiner

As is clearly shown, SOM training using a full dataset results in many redundant rules, located in the same region as the inefficiently described data (compared to the proposed method). The suggested modification of Gaussian MF gives a better description of the data in each rule. One can see that training the full dataset gives several redundant rules, as for example around $[10^6;1.25 \cdot 10^6]$ and $[10^6;0.25 \cdot 10^6]$. The same applies for other regions, and may degrade the understanding of classification decisions considerably. Otherwise, there may also be a need to limit the number of samples per SOM node in order to limit the number of fuzzy rules directly on the 1^{st} stage of NF.

Comparison to Other Machine Learning Methods

Most of the methods are implemented in publicly available tools libraries, such as WEKA, RAPIDMINER, LIBSVM, and DLIB. Yet some of them do not use parallel optimization, which can make the process of training extremely slow. This means that the amount of time required to train a model with several millions of data entries is enormous. However, MAPREDUCE can be used on the powerful stations to divide the tasks and process into parallel operations like was done using 12 nodes (12 threads each) by Triguero [417]. There have been several studies on the KDD Cup 1999 using a reduced set to train the model, though the HIGGS and SUSY datasets were barely used in the peer-reviewed publications due to their novelty and complexity. The accuracy results of WEKA 3.7.12 [149] (Bayes Classifier and C4.5) and DLIB 18.14 [6] (ν-SVM [91]) are presented in the Table 4.37. The results from the WEKA method need to be comparable to the results of the DLIB and our proposed method. At this point, we defined the Class 1 samples as *Negatives* and Class 2 as *Positives*. The 1^{st} class has $nC1$ and the 2^{nd} class has $nC2$ samples respectively out of a total N_S in the dataset. Using 5-folds cross validation in DLIB corresponding Sensitivity / True Positive Rate (Class 2 accuracy) and Specificity / True Negative Rate (Class 1 accuracy) were derived. In order to calculate the accuracy, we used the following formula:

$$Acc = \frac{TPR \cdot nC1 + TNR \cdot nC2}{N_S} \tag{4.14}$$

From the other side, WEKA provides a percentage of correctly classified samples in the results referred to as overall classifier accuracy.

The biggest challenge when using ν-SVM is estimation of the ν and γ parameters, which is mostly done by running cross-validation several times on the grid of values. However, each run of DLIB requires an enormous amount of time, so the values equal to 10^{-5} were chosen according to the library manual. The results of the proposed method and the existing best community-reviewed method

Table 4.37: Performance of other peer-reviewed methods on the defined datasets, including Soft Computing

Method	HIGGS	SUSY	RL	KDD	KDD_10
ν-SVM (dlib) - sens.	63.495	53.225	98.635	88.906	63.221
ν-SVM (dlib) - spec.	38.978	37.227	2.031	7.709	20.914
ν-SVM (dlib) - acc.	51.970	45.901	98.283	72.781	54.890
C4.5 (weka)	n/a	n/a	99.990	99.921	99.902
Random Tree (weka)	66.204	71.951	99.998	99.925	99.905
Bayes Network (weka)	66.356	75.854	99.983	99.414	99.121
M.layer (1) Perc. (weka)	65.004	78.451	99.998	95.271	94.689
M.layer (5) Perc. (weka)	67.484	78.896	99.998	95.492	94.944

are not that different, though the proposed method requires less time to learn from data and produces reasonable number of rules. Furthermore, we tried to run the LIBSVM implementation of $C-SVM$ from available in RAPIDMINER 6.3 [14]. However, the experiment did not finish within the reasonable time interval, so we considered it a failure. To compare with Hard Computing methods, on the KDD Cup 1999 10% dataset, C4.5 results in 171 rules, compared with 39 achieved by our proposed method with bootstrap aggregation. Another method was validated on Multilayer Perceptron with the result of 10 epochs, and 1 (5) hidden layer. Finally, we also used Random Tree, resulting in a tree with the size of up to a million units.

Note: This is that work can be beneficial for data analysis since existing tools like WEKA, GNUMERIC, and RAPIDMINER experience problems when dealing with data greater than 1 million samples.

Time Performance and Applicability in Live Systems

With the growing amount of network communications, the ability to mitigate attacks becomes crucial. Not only accuracy, but also the time required matters. For the proposed method, the learning and inference stages were measured, and an average required time is presented in the Table 4.38 to show the real time consumption for each of the processes. By improving upon the NF method, we achieved two goals: (i) fast training on large-scale datasets and (ii) the nearly real-time inference of a network packet in question. To compare to one of the fastest methods, Bayesian Network in WEKA took roughly 3 days to train from the HIGGS dataset, which was on an order of magnitude higher than the suggested method.

From the contemporary literature, we can see that Network Forensics Readiness

Table 4.38: Time in seconds required to learn models and infer new data for a different amount of fuzzy rules, using optimal SOM size without bootstrap aggregation

Dataset	Learning			Inference ,10^{-6}		
	Simple	Kosko	Proposed	Simple	Kosko	Proposed
HIGGS	938.38	940.95	5,816.36	2.96	2.90	26.00
SUSY	638.48	496.34	2,695.53	6.48	6.48	32.60
RL	238.01	198.61	1,781.29	3.10	2.74	13.90
KDD CUP 99	228.04	197.44	1,109.89	2.36	1.95	95.00
KDD CUP 99 10%	53.64	52.91	469.24	9.44	8.98	40.60

has many obstacles when dealing with wired and wireless communications in real world scenarios. Al-Mahrouiqi et al. [49] gave an insight into security awareness frameworks that included network forensics. Authors stated that it is important not only to collect and store log data properly, but also to maintain the knowledge warehouse. Furthermore, Adeyemi et al. [44] gave an overview of existing network forensics frameworks. Each of thee frameworks include complementary collection and analysis phases. We believe that fuzzy models for attack classification will help one understand attack scenarios better. Additionally, its applicability in live systems is feasible. From the Table 4.38, we see that it takes approximately 40 μseconds to classify a packet in question, in comparison to an average of 25,000 of network packets per second. We can see that KDD CUP was using much more additional field and meta data along with the standard fields (such as destination and source addresses, etc.) To compare with real systems, we looked into the study of network performance metrics given by Cisco[360]. According to the study, 100 Fast Ethernet links may give the following network packet rate PR depending on the packet size:

$$PR_{min\ size} = \frac{100 \cdot 10^6\ bits/s}{84\ bytes/packet \cdot 8\ bits/byte} \approx 148,809\ packets/s \quad (4.15)$$

$$PR_{max\ size} = \frac{100 \cdot 10^6\ bits/s}{1,538\ bytes/packet \cdot 8\ bits/byte} \approx 8,127\ packets/s \quad (4.16)$$

The suggested method works at 3 times faster processing speed than using a full packet in Fast Ethernet. However, by means of distributed sensors and multi-threaded implementation, one can achieve a suitable speed for Gigabit Ethernet networks. According to Internet bandwidth usage statistics [8], one can see that the highest average rate of connection was achieved in South Korea, and is equal to 26.7 Mbps. Thus, we believe that the proposed improvements make NF applicable not only for network forensics, but also for readiness and nearly real time incident

responses. Another concern is *incremental* or *online learning* that I defined by a faster model re-training when dealing with constantly changing data, as described by Bottou [83]. We believe that the proposed *offline* NF method can be transferred to the domain of online learning by means of two optimization procedures: (i) even faster SOM training 1^{st} stage using bootstrap on old datasets and newly arrived data over some fixed time frame; (ii) better stochastic optimization on the 2^{nd} stage that will reduce the number of required operations to tune fuzzy rules. In our view, this will provide a faster adjustment of existing the fuzzy rules parameters without retraining from scratch. One should consider the possibility however that new fuzzy patches may appear over time, resulting in a change of the fuzzy rules set.

Summary: We have proposed a new method for forensically-sound fuzzy rules construction using NF on the optimal SOM size determination using exploratory data analysis. The main idea is to produce an accurate model with a moderate amount of fuzzy rules that will also be human-understandable, and that can be used in Network Forensics Readiness effectively. The known Vesanto method for SOM size produces a vast amount of clusters that result in complex models, requiring enormous computational resources when dealing with Big Data in Digital Forensics Investigations, particularly with large-scale datasets that contain millions of data samples.

The proposed improvements of the first NF stage along with bootstrap aggregation gives a considerably better performance both in terms of accuracy and required training time. We have studied the influence of bootstrap aggregation on the final model using 1%. We can state that it can be used to improve the generalization of the extracted clusters, though the accuracy might degrade. Also, the Vesanto method is not suitable for use with bootstrap aggregation due to the lower number of samples available for clustering.

The new method for fuzzy patches and corresponding MF construction on the second NF stage improves classification accuracy. We can see that a lower number of fuzzy rules requires more advanced techniques to incorporate information from data, rather than using the Kosko method with triangular MF based on the projections. The proposed method for fuzzy patch construction uses χ^2 tests to find the parameters of each patch. Implementing the proposed method showed good accuracy and learning time reult on a desktop computer using four different datasets with an average amount of over 5 million samples. Current Machine Learning methods and their implementations are designed to handle tens of thousands of data, yet have complexity issues with bigger sets.

Big Data analytics requires new and enhanced models to handle complex prob-

lems such as Network Forensics Readiness and network traffic analysis. The conventional Neuro-Fuzzy method is very much affected by overfitting, the reduced explainability of the classification model, and the enormous training time required when it comes to million-sample sized datasets. As a result, faster and more accurate models are required. Therefore, the proposed improvements can serve as a stepping stone for real-time protection and forensically-sound evidence collection in network environments.

4.5.5 Overlap with Information Security Risk Management

Rowlingson [341] defined that Network Forensics Readiness and ISRM should be inseparable from better preparedness. Below, we consider the cases of malware distribution and DDoS attacks.

Case 1: Malware and Botnet Distributions

Successful malware distribution, such as in different versions of botnets, e.g. Zeus, Conficker[7] and others, have shown considerable resilience towards eradication. Epidemic models have proven useful for estimating propagation rates [450, 65], however historical data is more useful for obtaining probability distributions. We propose the following methodology for Malware and Botnet distributions:

1. *Data source.* For our calculations, we obtained data from the Shadowserver Foundation [8], which has monitored the infection rates of the Gameover Zeus botnet and Conficker with respect to time. Gameover Zeus is a peer-to-peer botnet built by cyber criminals, spread by sending emails with embedded malicious links or attachments, or enticing the victim to visit an infected website where a Trojan infects the victim. In comparison to APT statistics, the information about botnet distribution is relatively easy to gather from publicly available sources like Shadowserver because anti-virus companies construct corresponding signatures shortly after the first discovery of botnet and starts logging occurrences.

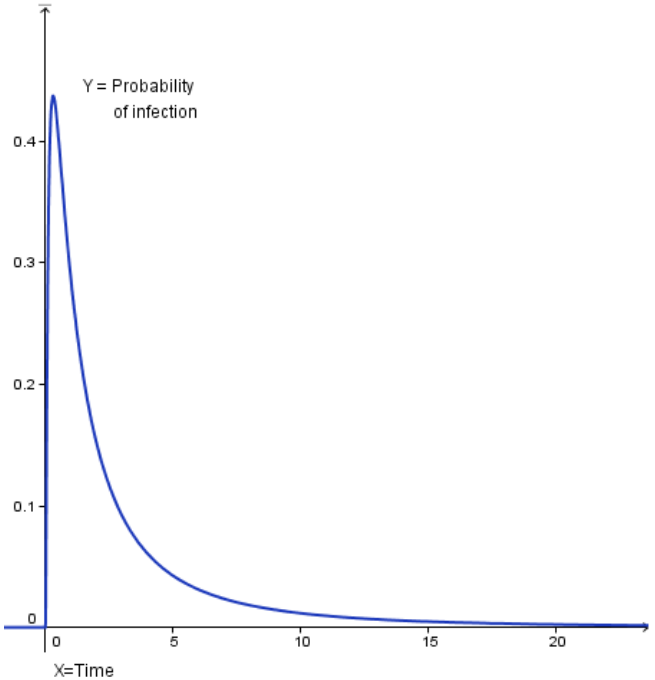
2. *Discussion of statistical approach.* Based on the available statistics collected over the months by Shadowserver, we ran a fitting test. The results concluded that the most promising hypothesis about the probabilistic model is that data follows the (1) LOGNORMAL distribution. Therefore, it can be possible to predict the exact percentage of probability of the distribution of the botnet in some future period. From the other side, numerical methods for

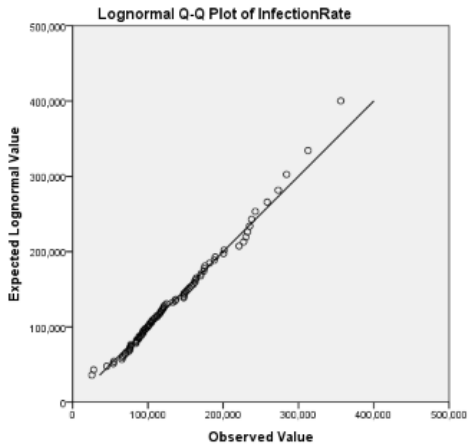
[7] Conficker was initially a computer worm, but when the payload was uploaded post-infection, it operated as a Botnet

[8] Gameover Zeus https://goz.shadowserver.org/stats/

time series analysis can estimate the number of malware species in the wild after a defined period. The value of the last two methods is that the trends of the malware distributions can be predicted with better accuracy than just random guessing, because human experts may fail to do it accurately.

3. *Results - Applicability of statistical methods and possible failures for each risk.* We can state that (1) the available data follows LOGNORMAL distribution, so we can use these methods to dicuss future conditions. (2) It is not possible to fully rely on these methods since the uncertainty in the predictions is quite significant due to versatility in the data and tail sensitivity in the graph. However, the derived information can be used in qualitative ISRM since it is a set of fuzzy metrics.





Estimated Distribution Parameters

		InfectionRate
Lognormal Distribution	Scale	120121.698
	Shape	.491

Figure 4.19: Gameover Zeus infection probability distribution and timeline. Right shows results of Q-Q plot of LogNormal distribution. Data source: The Shadowserver Foundation.

We ran the data points for Gameover Zeus in QQ plot and got the best fit with a Log-Normal curve, with a tendency towards a thick tail, Fig. 4.19. Our results show that the Gameover Zeus botnet distribution is left-skewed (positive). The initial propagation speed is high, until saturation or patch released slows down the propagation, from which point the existing population deteriorates. In addition to adhering to epidemic propagation theory, there are several aspects that will influence the thickness of the tail. For example, new versions of the malware being released, either exploiting a new vulnerability for increased propagation or changing behavior/coding to avoid scanners. In addition, we know that Conficker followed similar propagation and deterioration patterns, although Conficker[9] was self-replicating [450]. According to our model: if the entity is vulnerable, the general probability of infection is 30% from the initial dissemination until the first month has passed with a Mean population = 134,527, Standard Deviation = 64,797, and $\delta = 0.491$. The graph is sensitive to changes in the tails; this is also visible in the Q-Q plot results. The right tail of the graph in Fig. 4.19 would likely have been thicker if the data came from Conficker A+B, which remains active and deteriorates after six years.

4. *Classification of Risk* - Single non-zeroday malware infections are generally detected and removed by antivirus software, and generally pose very little risk. However, dependending on the target infected and type of malware, the payoff can be complex. Self-propagating malware is usually more severe, as they pose a threat to larger parts of the infected system. With some computer worms, the payoff can be considered simple, as the computer is infected (meaning non-operational) or not infected, effectively having only two states of being. It is partially possible to predict exposure from such generic attacks, e.g. the amount of vulnerable systems, but there is exposure to multi-vectored and other random effects which puts this risk in the *Third Quadrant*.

Case 2: Distributed Denial of Service (DDoS) Attacks

A denial of service (DoS) occurs when an ICT (Information and Communication Technology) resource becomes unavailable to its intended users. The attack scenario is to generate enough traffic to consume either all available bandwidth or to produce enough traffic on the server itself to prevent it from handling legitimate requests (resource exhaustion). The attacker needs to either exploit a vulnerable service protocol or to exploit a network device(s) to generate traffic, or to amp-

[9] See also http://www.shadowserver.org/wiki/pmwiki.php/Stats/Conficker

lify his requests via a server to consume all of the bandwidth. The DoS attack is distributed (DDoS) when the attacker manages to send traffic from multiple vulnerable devices. The attacker can achieve amplification through the exploitation of vulnerable protocols or through using botnets.

The increase of Internet throughput capacity has also facilitated the growth in traffic volume for DDoS-attacks. According to Arbor Networks, the largest observed attack in 2002 was less 1 Gbps (Gigabit per second). While the biggest observed attack until now targeted a British television channel and reportedly generated ≈ 600 Gbps of traffic. That is an approximate 60x development in capacity for DDoS attacks over the course of about 14 years, see Fig. 4.20.

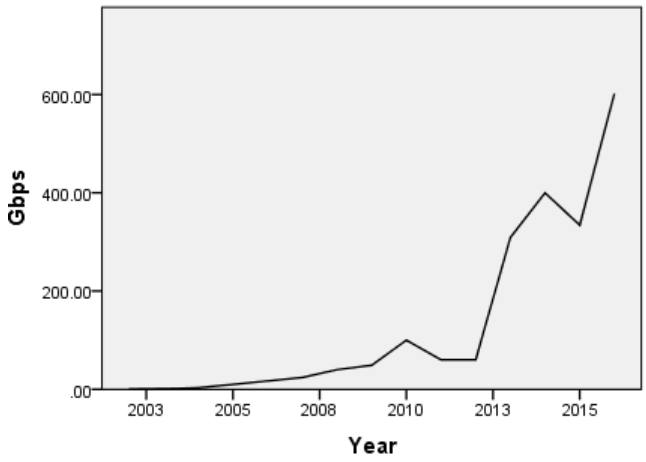


Figure 4.20: The development of bandwidth consumption (Gbps) of DDoS-attacks during the last 15 years. *Data source: Arbor Networks and media reports*

One of the most feared information attacks is the DDoS attack, as they have the potential to break servers and deny access to a service to customers over an extended period, causing massive revenue losses. By monitoring activity, we can obtain reliable numbers on how large the average DDoS attack is, and generate distributions of attack magnitudes.

1. *Data source.* There are open access statistics available on DDOS attacks. So, we can use available statistics, yet it can not be fully relied on due to misleading detections or hardware malfunctions. Using numbers gathered from open access, we generated an example of the possible distribution of DDOS occurrences for different bandwidths, as shown in the Figure 4.23. Available threat intelligence indicates that the commonly observed DDoS

magnitude at the time was between 0-90 Gbps, with distributions as seen in Table 4.39. Our test dataset corresponded to the numbers provided in open access sources, having an arithmetic mean = 7.31, and Std. Dev = 13.55. The largest reported DDoS attack so far was 500 Gbps, and we can estimate that the generic probability of such an attack occurring annually is large; while the probability of such a large-scale directed attack at a single organization is negligible. There was no observed attack magnitudes over 90 Gbps in the surveys. However, we add the scenario A5 in Table 4.39.

Table 4.39: Example of DDoS attack magnitude distributions and probabilities, with conditional probabilities of semi-annual occurrence.

Scenario	Gbps	% of attacks	P(A|B)
A1	<1	55.00 %	27.50%
A2	1-5	15.00 %	7.50%
A3	5-10	10.00 %	5.00%
A4	10-90	20.00 %	10.00%
A5	90+	Not observed (0.1%)	0.05%

2. *Discussion of statistical approach.* There are several possible ways of approaching the statistical analysis of DDOS attacks. At first, the probability of the DDOS attack can be calculated as simple (1) CONDITIONAL PROBABILITY, which gives an exact risk of being targeted for a DDOS attack out of possible attacks. Table 4.23 shows the results of calculations made for an organization that expects a P(B)=50% annual chance of DDoS attack. After that, we can say something about the number of attacks and maximum possible use of bandwidth by considering historical information. However, the number of maximum reported DDOS attacks follow the (2) EXPONENTIAL FUNCTION and can not be predicted for following years: $N = N_0 \cdot e^{t'}$ since some covert parameters are not taken into consideration like breakthrough network controller speed. Finally, the particular scenario involving discretion intervals of DDOS bandwidth are considered, such as $P(DDOS > 90Gbps) = P(DDOS) \cdot P(> 90Gbps|DDOS)$. Also, the (3) γ-DISTR. is the most applicable way of modeling such variety in the scenarios.

   Fig. 4.21 depicts the correlation between duration and magnitude, where the attacks from the *A1* and *A2* scenarios are distributed nearly uniformly across the duration scale. It means that the nature of such attacks is more random and non-deterministic, which was also confirmed by our correlation tests. Going further, one can see that the majority of the attacks from the range of *A3* are located in the duration range around $10^3 \cdots 10^6$ seconds. Finally, the same stands for the scenario *A4*, where the dispersion of possible

magnitudes is large in comparison to *A3*. However, date with much higher frequency in the case of the probabilistic model suppresses less frequent cases, while fuzzy logic describes data independently from the frequency of its appearance, only taking into consideration its possibility as described before by Shalaginov et al. [373].

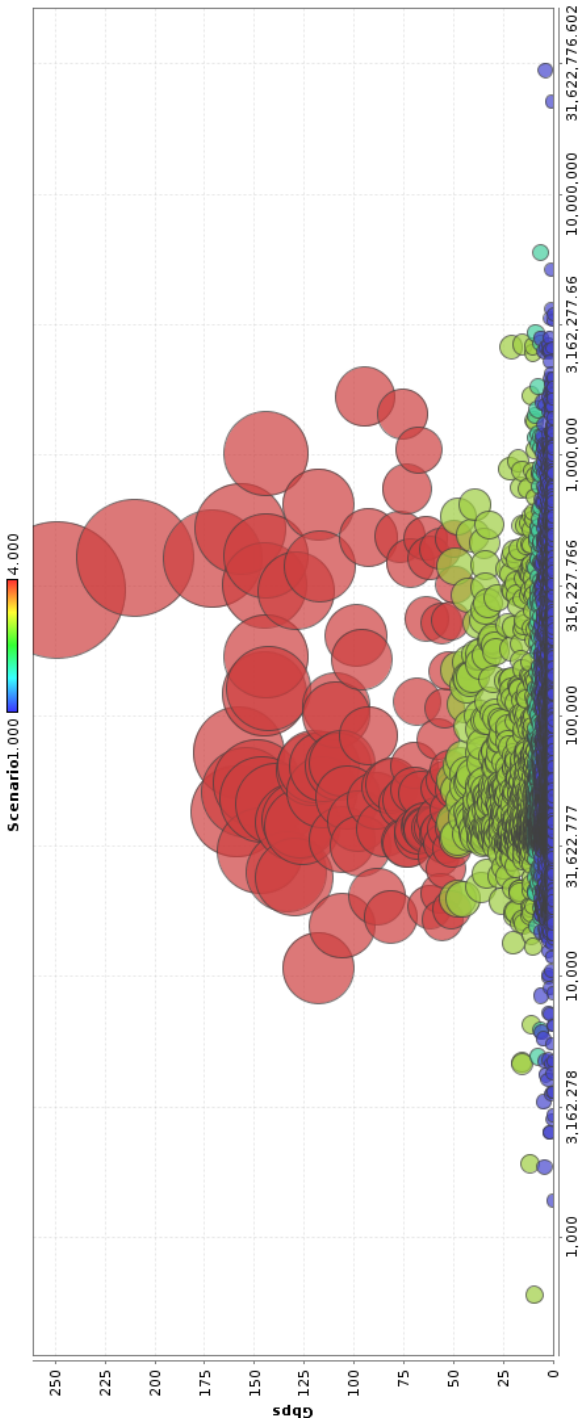


Figure 4.21: Bubble plot of the attack bandwidth depending on the duration for each scenario. The size of the bubble also denotes the magnitude of the attack. Scenarios are depicted with different colours.

Probabilistic modelling for Risk Estimation. Unplanned downtime is an adverse event for which most ICT-dependent organizations need to have contingencies. The institution considered in this paper has defined the severity metrics in Table 4.40 as ranging from "Negligible" to "Catastrophe", together with the distribution of durations within the defined intervals. Losses are considered to be moderate up to two hours downtime, as most employees will be able to conduct tasks that do not require connectivity for a short period. Losses are estimated to start accumulating after 2 hours of downtime. The analysis shows that the defined events *B3-B5* are over 99% likely to last more than 2 hours, which falls well outside of the Institution's risk tolerance.

Table 4.40: Overview of attack severity for the case study and duration frequencies. *Data Source: Akamai [31]*

Outcome	Interval (min)	Seconds	Severity	Frequency	% of Attacks
B1	0-10 min	0 - 600	Negligible	1	0.0
B2	11-30 min	601 - 1,800	Slight	1	0.0
B3	31 - 120 min	1,800 - 7,200	Moderate	28	0.6
B4	2 - 24 hours	7,201 - 86,400	Critical	3,346	70.2
B5	>24 hours	> 86400	Catastrophe	1,392	29.2

Possibilistic modelling. Fuzzy hierarchical models represent possibilities modelling that differ from the conventional probabilistic approach for Risk Assessment. Our main motivations are (i) to eliminate negligibly low probabilities when an event is near, (ii) to apply inexact fuzzy rules for better generalization of risk assessment model and (iii) to compose a hierarchical model that covers all the components of risk assessment in a single framework.

In order to be able to fuzzify the earlier defined scenarios, we performed a proof-of-concept demonstration of fuzzification and corresponding fuzzy rules extraction. We used the Neuro-Fuzzy method for automated data analysis as earlier defined by Shalaginov et al. [373]. The method analysed both attributes *Duration* and *Magnitude* and proposed the 12 most suitable fuzzy rules for four scenarios as defined by us earlier. The distribution of the rules is shown in the Figure 4.22 and plotted with the help of RAPIDMINER [14]. The biggest bubbles denote centers of extracted fuzzy rules. The MF of the fuzzy rule in the center has a value of 1.0, decrease around it according to values of standard deviation for each of the attributes. One can see that the extracted rules for the scenarios *A2* and *A3* located nearby and have similar statistical parameters. On the other hand, rules for scenarios *A1* and *A4* do

not overlap and have quite distinct placements on the scatterplot. We are able to classify up to 99% of the DDoS attacks using this fuzzy model based on two attributes.

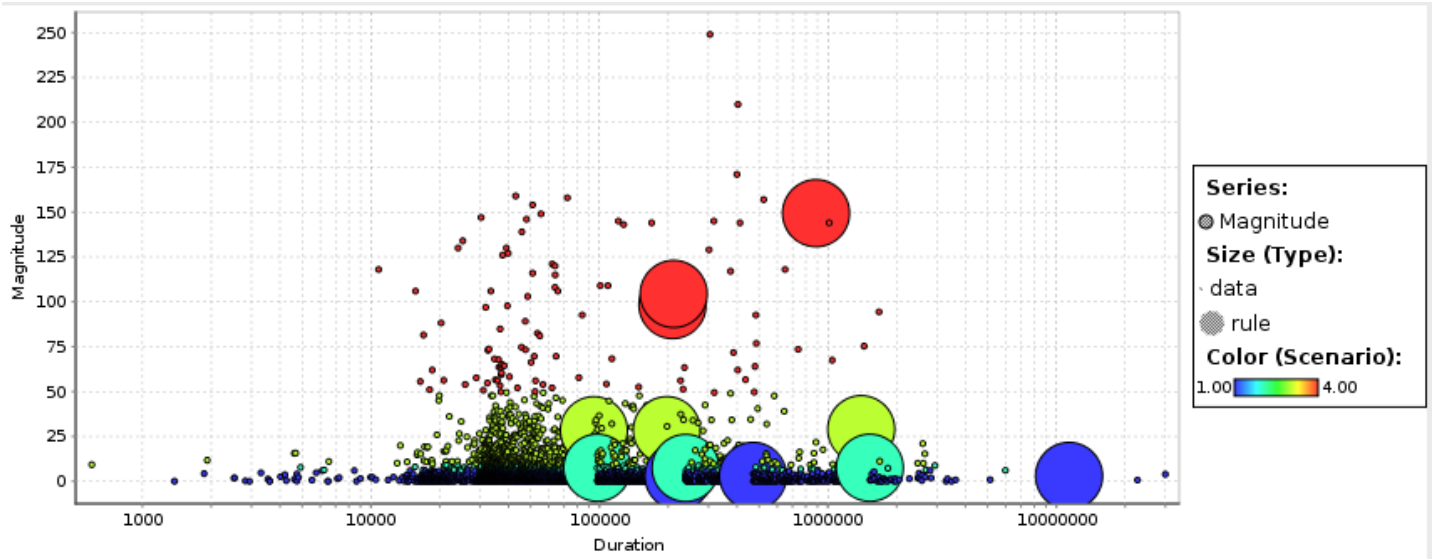


Figure 4.22: Distribution of 12 fuzzy rules extracted automatically with respect to data location. Centers of extracted fuzzy rules are depicted with big bubble;, original data points with small ones.

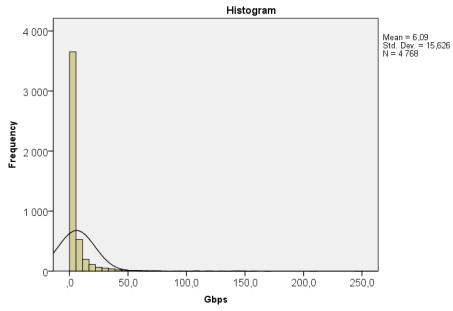
The advantage of the Neuro-Fuzzy method is that it is suitable for large-scale data analysis when the number of attributes is much greater than two. Moreover, it gives us a clear understanding of the properties of similar attacks for each scenario, even when their nature is non-deterministic and does not have clear a dependency between attributes, like in case with *A1* and *A2*.

Furthermore, we can perform a manual analysis of the data in order to fuzzify them using manual analysis. As we described the in the methodology, the corresponding parameters of the fuzzy patches should be extracted. We use Gaussian MF that is composed of the *mean* of the corresponding scenario magnitude distribution with *standard deviation*. The results are given in the Table 4.41.

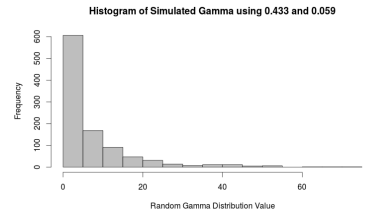
Table 4.41: Parameters extracted from different scenarios for Gaussian MF

Scenario	Mean	Std.Dev
A1	1.4351	1.5318
A2	7.2954	0.8429
A3	19.2653	9.7245
A4	92.7211	39.541

Using these parameters, we compose four MF for each of the ranges of the DDOS attacks magnitudes. Also, results of the modeling in the Figure 4.23 show that the distribution of DDOS attacks magnitude follows γ distribution with the following parameters: *shape*=0.152, *scale*=0.025.



(a) Histogram of DDoS magnitude displaying normal curve



(b) Simulated γ-distribution of DDOS from collected data

Figure 4.23: Comparison of the original DDOS data and modeled distribution

Moreover, to compare the probabilistic modeling with fuzzy logic reasoning, we plotted both approaches in the Figure 4.24 using R[161]. One can see

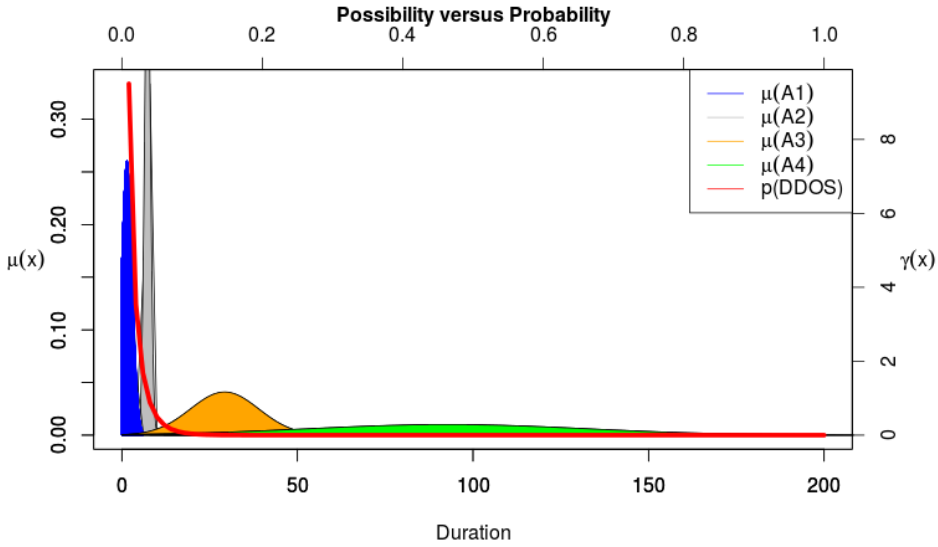


Figure 4.24: Mapping fuzzy logic-based Gaussian MF μ and probabilistic density function of γ distribution

that the main difference from the probabilistic approach is the independence of the possibility of a number of particular event occurrences. This means that fuzzy logic can help to predict a particular outcome using inexact linguistic terms with respect to how far the numerical value lies from 100% degree of truth of the corresponding linguistic term.

Fuzzy logic has great utility in uncertainty modelling when there is not enough data for probability estimation. Insufficient data for probability will cause very low numbers increasing possible calculation errors. For example, in the DDoS example above, one can see that with a growing attack duration, the value of the γ density function will be millions in parts of 1.0. The Figure 4.24 gives insight into the fundamental difference between two modeling principles. The *theory of possibility* by Zadeh [454] was proposed as an alternative to the *theory of probability* for showing the uncertainty also can be described by the degree of trust of a particular statement rather than the chance of its occurrence. Thus, at this point it crucial to note that the conventional statement *P (Occurrence for organization x Magnitude x duration)* may be transferred to fuzzy logic inference:

$$\gamma(X) \Rightarrow \mu(X) \tag{4.17}$$

By introducing fuzzy logic, we believe that IRSA can not only reduce un-

certainty caused by extreme events, yet also bridge between raw numerical characteristics and the linguistic assessment provided by human experts. Also, from the Akamai case above we can see that a negligibly low probability does not necessarily properly describe the series of events that fall into the scenario A4. Moreover, the fuzzy logic model results in a set of rules that can be used by decision makers.

3. *Results - Uncertainty/ Confidence intervals.* The data and estimated parameters are valid only for some period until new attack methods emerge. However, it is still possible to form a corresponding γ-distribution to characterize the bandwidth for DDOS as it is depicted in the Figure 4.23, (a). Thus, the corresponding CI can be extracted based on the parameters of the distribution to estimate the DDOS [207]. The Lower boundary can be neglected, however exceeding the upper boundary may indicate that the parameters need to be re-evaluated for quantitative ISRM.

   Though the data distribution can vary, and therefore change the form depending on newly emergent technologies in the network adapters industry, we can still use CI to estimate the boundary of the desired mitigation frame. It can be stated that the company wants to eliminate some % of the DDOS attacks and estimates the threshold of the attacks based on previously collected information. The Table 4.42 presents an exact range of the bandwidth at which a particular % of attacks can be mitigated. Our particular interest is in the upper boundary of the CI, since the lower boundary can be ignored at this point. For example, to withstand 95% of the DDOS attacks according to the modeled γ-DIST. in the Fig. 4.23, (b) a company must implement a DDOS protection not lower than 62.82 Gpbs.

Table 4.42: Confidence Intervals for defined % of the DDOS attacks to be eliminated

To eliminate	50%	90%	95%	99%
Limit_lower, Gbps	0.531143	0.012634	0.002547	0.000061
Limit_upper, Gbps	9.411601	29.566104	39.241385	62.822911

4. *Results - Applicability of statistical methods and possible failures for each risk.* We can estimate and establish a threshold for intrusion detection systems to be capable of handling such attacks, which might be significant when estimating the risk that the organization takes when ignoring particularly intensive attacks. For example, network adapters have increased capacity from 100Mbps up to 1Gbps over previous years. Therefore, statistical models can be used for (1) DDOS bandwidth and probability prediction and estimation, though constant failures of these models may indicate a need

for re-evaluation of the maximal DDOS bandwidth. Furthermore, using the estimated probability, we can also build qualitative risk estimators as more general linguistic characterizations of the risk.

5. *Classification of Risk* - As we have shown, it is possible to obtain distributions of DDoS attack magnitudes with their associated probabilities. Our observations can be offset however by a single massive attack, such as DDoS attack on Estonia in 2007 attributed to Russia. This area is also subject to Moore's law, which means that the historical observations of attack magnitudes will quickly become obsolete. We consider the payoff from DDoS attacks as simple; it either succeeds in denying service or it does not, while the duration of the attack determines the consequence. Our analysis, therefore, places the risks of DDoS attacks in the *Third Quadrant*.

4.6 Mobile-Device Virus Analysis[10]

Over the last decade, one can observe a malware transition from Personal Computers to mobile platforms that is caused by the intensive development of embedded devices. Despite the fact that, there has been a number of awareness campaigns (Europol [138]) about the amount of malware in the environment, and affected users are growing as shown in reports from 2017 [399, 421].

4.6.1 Datasets

The proposed improvements to the Neuro-Fuzzy rules-extraction classification method were tested on the **Android malware dataset**. Moreover, there have been several datasets chosen to demonstrate the proof of concept from the UCI Machine Learning Repository [16]. Besides the properties of the datasets presented in the Table 4.43, the ratio of the biggest eigenvalues $\frac{e_0}{e_1}$ from a non-zero mean set (Correlation Matrix) and absolute averaged Person Correlation Coefficient $|\bar{r}|$ are given.

The other datasets are publicly available and represent different domains. All datasets chosen are with two classes, as in the malware detection problem. The *mobile malware* is a manually composed dataset of features from static and dynamic tests of mobile *malware* and *benign* applications. The Android platform was chosen because of its popularity, since users can install 3^{rd} party applications (not from official Google Play) that might contain malicious payloads. We got 388 unique malware samples with only 348 suitable for dynamic and static tests. Furthermore, 460 unique applications (247 suitable for testing) known to be benign applications

[10]The main ideas of this section are published under the contributions [368, 371, 373, 374]

Table 4.43: The properties of the datasets used in the experiments based on the data, obtained from the statistical programs PSPP [311] and Weka [149]

Dataset	M	N_S	**Const.**	**Norm.**	$\bar{\lvert r \rvert}$	e_0	e_1	$\frac{e_0}{e_1}$
Climate Model Crashes	18	540	No	Yes	0.0091	1.0797	1.0759	1.0035
Fertility	9	100	No	Yes	0.1204	1.8654	1.4530	1.2838
Banknote Authentication	4	1,372	No	No	0.4255	2.1799	1.2931	1.6857
Mobile malware	36	596	Yes	No	0.1224	6.3839	4.3715	1.4603
Ionosphere	34	351	No	Yes	0.2169	8.8121	4.2386	2.0790
SPECTF Heart	44	80	No	No	0.2516	11.9965	6.3279	1.8957
Madelon	500	2,000	No	No	0.0182	6.4630	5.0242	1.2863
QSAR biodegradation	41	1,055	No	No	0.1653	7.3993	5.0285	1.4714

were gathered from official sources. In this dataset, we assigned *Class 1* to a malicious application's features vector and *Class 0* to corresponding vectors of benign application. Finally, it consists of 36 versatile numerical features from 596 applications that characterize a particular property such as CPU load, memory usage, API calls, etc., as also described in [371]. The specification of the features is given in the Table 4.44.

In the Figure 4.25, the visualization of the dependency between the features in the given earlier datasets is shown. In the cases of independent features, the number of fuzzy patches can be decreased by covering more instances. From the other side, some non-monotonic dependencies require an increase in the number of fuzzy patches to reduce error.

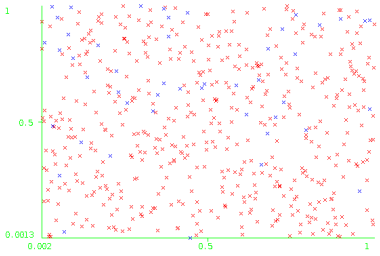
From the scatter plots of selected pair-wise attributes, we can see that the characteristics of dependency between attributes vary a lot. At this point, we can expect that the *Banknote Authentication* dataset might require the largest number of more specific rules to build the classification model, considering the non-linear and non-monotonic dependencies in the Figure 4.25.c. Furthermore, similar challenges may face rules-generation for the *Ionosphere* dataset 4.25.e and *QSAR* dataset 4.25.h.

4.6.2 Experimental Design

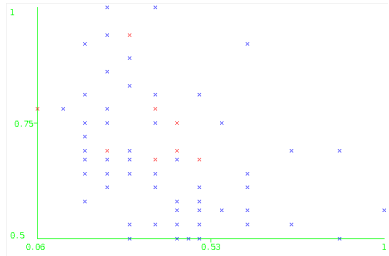
To evaluate the performance of the proposed scheme, we implemented two stages of the NF method as described by [233]. Also, we used three types of the fuzzy patches and corresponding MFs. The first one is *simple rectangular patches with triangular MF*. The second one is [233] *elliptic patches with projection-based triangular MF* based on the projection of the ellipses. The third is *elliptic patches with modified Gaussian MF*, proposed by [374]. Since SOM is a method influenced by the random initialization of the nodes, bootstrap aggregation was performed. To ensure the consistency of the results, 10 samples were randomly gen-

Feature name	Description
sdkVersion	Minimal version Android API required for app execution
targetSdkVersion	Target version of Android API necessary for app running
app_label_length	Length of the application label
package_name_length	Length of the package label, e.g. 'com.application'
filesize	Size of installation package file
permissions_highest	The highest numerical level of permissions
permissions_avg	Average numerical level of permission
permissions_number	Amount of permissions requested by application
pull_data_size	Amount of stored data in '/data/data/<app>'
log_launch_size	Size of pulled log file during app launch
cpu_usage_peak	Peak value of CPU utilization during during launch
cpu_usage_avg	Average value of CPU utilization during launch
cpu_usage_stdev	Standard deviation value of CPU utilization
thr_usage_peak	Maximal amount of created threads
thr_usage_avg	Average amount of threads created by an app
thr_usage_stdev	Standard deviation of amount of threads
vss_usage_peak	Maximal size of virtual memory used by an application
vss_usage_avg	Average size of virtual memory used by an application
vss_usage_stdev	Standard deviation of size of virtual memory
rss_usage_peak	Maximal size of resident memory used by an app
rss_usage_avg	Average size of resident memory used by an app
rss_usage_stdev	Standard deviation of size of resident memory
shared_prefs	N of XML files in '/data/data/<app>/share_prefs'
shared_prefs_size	Size of all shared preferences files
databases	N of stored DB in '/data/data/<app>/databases'
databases_size	Size of all stored databases
files	N of stored files in '/data/data/<app>/files'
files_size	Size of all stored files
package_entropy	Shannon Entropy, which helps to reveal encrypted info
package_number_files	Total number of files in the installation package
manifest_size	Size of configuration file 'AndroidManifest.xml'
res_folder_size	Size of the folder with additional resources
assets_folder_size	Size of directory with application's assets
classes_dex_size	Size of DEX files (already compiled Java classes)
classes_Dex_entropy	Shannon entropy of DEX files
execution_time	Time that was spent to execute app

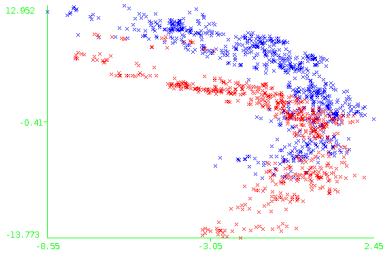
Table 4.44: Example of collected features in mobile malware dataset



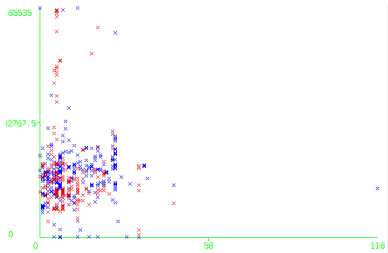
(a) Climate: vconst_4 vs convect_corr, r=-0.01



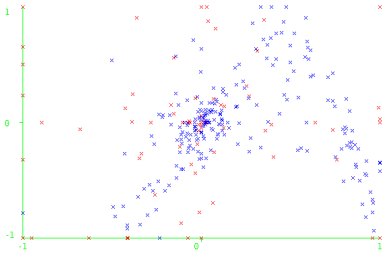
(b) Fertility, Age vs Sitting hours, r=-0.44



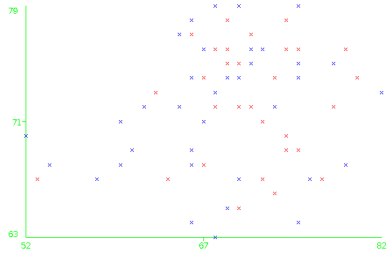
(c) Banknote: Skewness of wavelet vs Entropy o fimage, r=-0.53



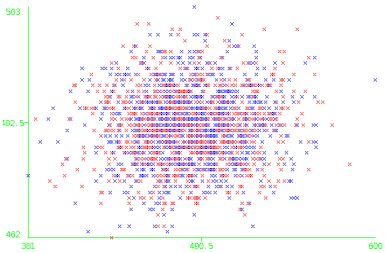
(d) Mobile: permissions_number vs log_launch_size, r=0.01



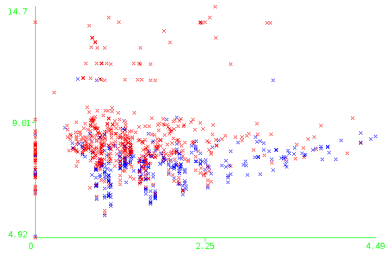
(e) Ionosphere: a12 vs a32, r=0.47



(f) SPECTF: F2R vs F4R, r=0.2



(g) Madelon: a1 vs a2, r=0.0



(h) QSAR: *LOC* vs *SM6_B(m)*, r=-0.01

Figure 4.25: Visualization of the dependencies between the features in all datasets mentioned earlier in Weka with corresponding values of PCC. The blue and red colors denote classes

erated during SOM clustering and the best sequence was chosen to be used afterwards in the experiments. On the 1_{st} stage of NF, the fuzzy patches parameters are derived according to the derived number of clusters from the proposed method. N_S training epochs were used in SOM. During the 2^{nd} stage, the classification rules are tuned. We used 100 training epochs for 2^{nd} stage of NF and a 0.1 training rate for both stages.

4.6.3 Performance Evaluation

The accuracy of the NF method was evaluated using two different performance metrics: (1) a regression-based real-value accuracy comparison of the defuzzified output of NF using the following metric: Mean Absolute Percent Error $MAPE = \frac{1}{N_S}\sum_{i=1}^{N_S}|\frac{y_i-d_i}{d_i}| \cdot 100\%$, where y_i - the output of of the gravity defuzzifier [233], d_i - the actual class of the sample, and N_S - number of given data samples, and (2) Discrete classification accuracy comparison using the selected rules by min-max principle in cross-validation: $Acc = \frac{N_P}{N_S}$, where N_P - number of properly classified samples.

4.6.4 Results & Analysis

The hypothesis was made that a greater size of SOM provides more unstable results due to the spread of data. A trade-off must be found in order to reduce complexity and improve specificity. To evaluate the defined hypothesis that a greater size of SOM gives more unstable results due to the spread of data, we performed a comparison of cross-validated results with bootstrapped training data.

Effect of Self Organizing Map Size on Classification Accuracy

The performance was evaluated using the regression performance metric "MAPE, %" and a classification one "Acc(rules), %". The results are presented in the Table 4.45 for all the given dataset. The *theoretical number* of rules in the Table means the number of rules suggested by SOM size estimation methods independently from the number of classes. The *extracted rules* means a number of rules actually extracted, also considering different classes, implying that the second measure can exceed the first one. In addition to this, we performed a comparative study of manually-defined SOM sizes and corresponding classification results which are given in the B. One can see that proposed procedure for optimal SOM size determination results in a high classification accuracy and less complex model.

The proposed method shows how SOM size along with Gaussian MF seems to fit well with the data. Also, classification accuracy using the *min-max* principle always exceeds Kosko method. Using the χ^2 for building fuzzy patches provides a better way to fit data. Thus, we can summarize that both original models do not

Table 4.45: Performance comparison (regression, classification) of the proposed method

Method	"Rule of thumb"		Vesanto		Proposed	
	MAPE,%	**Acc**,%	**MAPE**,%	**Acc** ,%	**MAPE**,%	**Acc** ,%
Dataset: Climate Model Simulation Crashes Data Set						
Simple MF	13.3848	23.5185	13.3825	23.5185	38.3623	22.7778
Kosko MF	11.1754	**91.4815**	45.4244	**91.6667**	45.6569	92.0370
Gaussian MF	5.9791	69.6296	5.8009	76.6667	3.8818	**96.4815**
Extr.rules (Theoret.)	**61 (116)**		**61 (117)**		**6 (6)**	
Dataset: Fertility						
Simple MF	14.7705	88.0000	18.5313	88.0000	68.8893	84.0000
Kosko MF	6.0117	88.0000	10.2212	85.0000	8.9175	89.0000
Gaussian MF	6.2184	**92.0000**	6.5119	**92.0000**	4.8512	**92.0000**
Extr.rules (Theoret.)	**9 (50)**		**9 (64)**		**7 (11)**	
Dataset: Banknote Authentication						
Simple MF	22.2296	55.5394	21.8893	55.5394	5.9234	95.5539
Kosko MF	27.1998	98.9796	22.4358	98.9067	22.7381	96.2828
Gaussian MF	1.7491	**100.0000**	15.7715	**99.6356**	18.0398	**99.9271**
Extr.rules (Theoret.)	**94 (185)**		**97 (312)**		**17 (25)**	
Dataset: Mobile malware						
Simple MF	38.4787	35.1261	39.0162	40.0000	40.1554	41.1765
Kosko MF	41.5120	58.4874	34.5776	56.8067	41.3487	58.4874
Gaussian MF	11.7330	**84.5378**	11.5823	**80.8403**	5.1865	**91.9328**
Extr.rules (Theoret.)	**48 (122)**		**39 (189)**		**24 (13)**	
Dataset: Ionosphere						
Simple MF	44.9931	26.2108	39.2309	29.6296	25.2921	64.1026
Kosko MF	43.4663	48.7179	12.5893	49.5726	44.8586	49.5726
Gaussian MF	23.2906	**80.9117**	22.5460	**74.0741**	15.5957	**88.3191**
Extr.rules (Theoret.)	**26 (94)**		**22 (195)**		**28 (22)**	
Dataset: SPECTF Heart						
Simple MF	66.3703	33.7500	67.6426	31.2500	40.7887	41.2500
Kosko MF	25.0000	72.5000	25.0000	75.0000	33.6323	72.5000
Gaussian MF	16.8750	**85.0000**	16.5625	**86.2500**	13.3333	**85.0000**
Extr.rules (Theoret.)	**2 (45)**		**2(89)**		**3(24)**	
Dataset: Madelon						
Simple MF	37.1836	49.5500	37.5393	49.1500	44.8408	41.1500
Kosko MF	37.2098	49.9500	25.3145	49.8000	26.9802	61.9000
Gaussian MF	42.2500	**78.7000**	44.2000	**77.7500**	36.1000	**82.1500**
Extr.rules (Theoret.)	**197 (224)**		**215 (288)**		**6 (5)**	
Dataset: QSAR biodegradation						
Simple MF	25.7192	46.2559	26.8668	48.8152	28.6848	45.6872
Kosko MF	26.7484	67.4882	26.7991	69.0047	29.4705	68.1517
Gaussian MF	13.1149	**75.9242**	15.0202	**73.9336**	10.4234	**87.2038**
Extr.rules (Theoret.)	**101 (162)**		**99 (239)**		**23 (14)**	

possess the same accuracy as the proposed one, neither in regression performance nor in classification based on the fuzzy rules model.

The results for all 8 datasets look consistent. We can draw the conclusion that the proposed method tends to decrease the number of extracted rules significantly while accuracy consistently decreased. Also, we improved Gaussian MF by [373] describing data within the fuzzy patch much better than when using either simple triangular MF or the projection-based triangular MF introduced by Kosko. *SPECTF* dataset can be described by using only 2-3 rules due to the lower number of rules and more random distribution of the features values. However, the accuracy of fuzzy model based on the proposed method is 85% using 3 rules, while rectangular fuzzy patches with triangular MF gave only $\approx 30\%$ properly classified samples. Further, *"Rule of thumb"* gives 100% classification rate for the *banknote authentication* dataset, which is 0.0779% more than for the proposed method. Yet the number of rules is 94 versus 17, which also implies the trade-off between the complexity and accuracy of the model is better in the proposed method. The only irregularity that was noticed in the results is related to the *Climate* dataset, where the Kosko method gives better results than the *Vesanto* and *"Rule of thumb"* methods, yet not on the proposed one.

Influence of the Proposed Method on 2^{nd} Stage of Neuro-Fuzzy Training

The size of the SOM depends, first of all on the statistical properties of the data set. We can see that in most cases the results are degraded with the increase of the SOM grid. However, we can observe the results for the rectangular patches and Kosko patches on the datasets with non-normalized and correlated features. Despite this, the proposed method performs well on such datasets with a considerable amount of features keeping the classification accuracy of around 90%. To see how well the patches describe the data, the MAE (Mean Absolute Error) was observed during the learning on the 2^{nd} step of NF. The results for all three methods are given in the Figure 4.26.

As can be seen, the rules are mostly tuned based on the rectangular Kosko method with corresponding triangular MF during the 2^{nd} training phase. In contrast, rules based on the proposed methods give the best results with almost no alteration of the weights during that training. Therefore, we can state that the proposed patches fit the data with the lowest possible error. It also results in drastically reduced training overhead on the 2^{nd} stage of NF.

Allocation of the Fuzzy Rules and Compactness of Self Organizing Map Nodes

To understand the difference between the three methods in terms of fuzzy patches allocation we decided to visualise the way these patches are located. Since it is

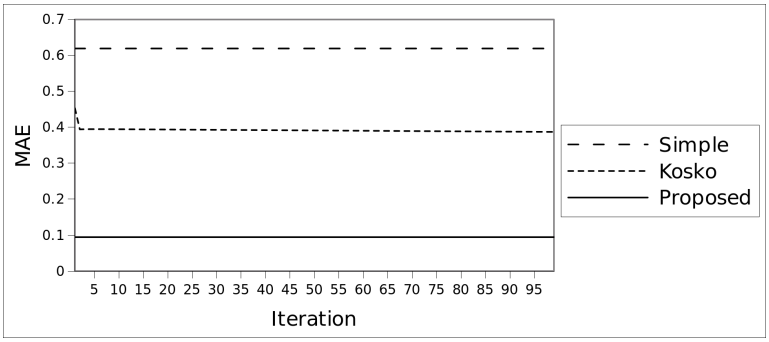


Figure 4.26: Change of MAE values of three methods on the 2^{nd} stage of NF on *mobile malware* dataset over training with 100 epochs

not possible to visualize all 41 features for the *Mobile malware* dataset, we only took the two features that contribute mostly to the classification according to ReliefF merit. These are the 2^{nd} and 6^{th} attributes, called "target SKD version" and "highest request permission" respectively. The results of the visualization (scatter plot) of these two attributes are given in the Figure 4.27 using RapidMiner [14]. The shape of the points indicates class label, while the color presents a method used to determine size of the SOM.

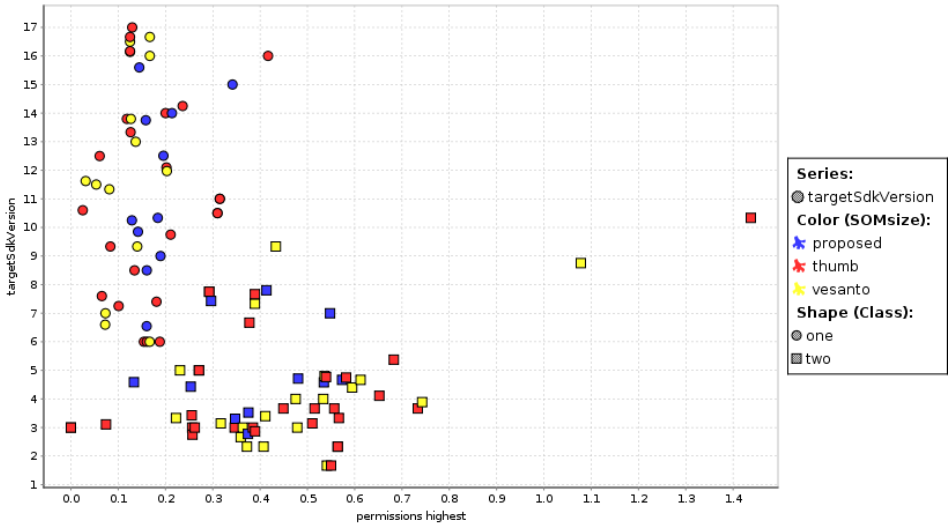
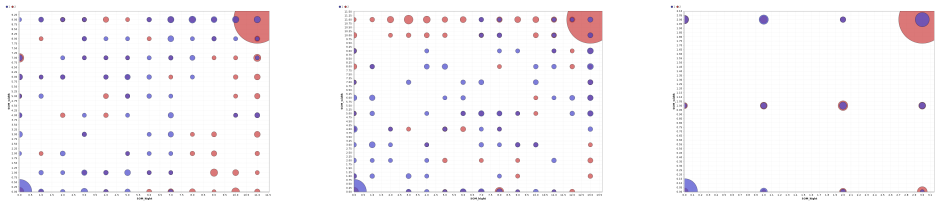


Figure 4.27: Distribution of the fuzzy rules derived based on three different SOM size estimation methods

We can see that *"Rule of thumb"* and *Vesanto* methods result in a number of redund-

ant rules, which located in the same region also describing data in an inefficient way comparing to the proposed. Also, the usage of Gaussian MF gives a better description of the data in each rule. Around the region [0.2;16], one can see 3 rules for *Vesanto* and *"Rule of thumb"*, while for the proposed method it gives only a single rule for the same class number 1. The same applies for other regions that may improve the understanding of classification decisions.

One can see an explanation through the allocation of data samples from different classes in SOM nodes, as depicted in the Figure 4.28. The plot is a bubble plot built using RapidMiner [14]. Despite the fact that the SOM sizes for Vesanto and "rule of thumb" methods are not large, there still exist a dozen empty nodes. In addition to this, one can see fewer nodes that contain a majority of the assigned data samples. While the proposed method offers a lower number of SOM nodes with a higher accuracy, using a manually defined SOM grid sized in the B, we can see that many datasets give a majority of empty nodes causing much computational overhead.



(a) "Rule of thumb", 9x11 **(b)** Vesanto, 11x13 **(c)** Proposed, 2x3

Figure 4.28: Distribution of date samples per SOM node with respect to different classes using three size determination methods. The size of the bubble corresponds to number of samples in this node, while colour denotes malicious or benign sample

Performance using Other Machine Learning Methods

In order to reliably establish the relevance of the proposed improvements to the NF, comparisons to other community-approved ML methods are given in the Table 4.46.

Note: we used Weka package with 5-fold cross validation. Also, ANN was launched with 3 hidden layers. All methods have weak performance on *Madelon* on a range $\approx 50 - 65\%$, while the proposed method shows accuracy up to 82%. This is a significant difference, which can be explained by the need for generalization on datasets with hundreds of features. On the other datasets we can see results consistent with the proposed method, as described in the Table 4.45.

Table 4.46: Accuracy of the other ML methods on the datasets. Highest accuracy is denoted with bold.

Dataset	Bayes Network	ANN	SVM	C4.5
Climate Sim.Crash.	92.2222	**93.1481**	91.2963	91.8519
Fertility	**88.0000**	84.0000	**88.0000**	84.0000
Banknote Auth.	91.9825	**100.0000**	**100.0000**	98.6152
Mob. malw.	90.5882	92.6050	58.4874	**93.2773**
Ionosphere	89.4587	90.5983	**92.5926**	89.7436
SPECTF Heart	72.5000	**75.0000**	53.7500	**75.0000**
Madelon	60.6500	57.5500	50.0000	**64.9500**
QSAR	80.4739	83.6019	**85.8768**	84.3602

4.6.5 Complexity

It can be noted from the theoretical justification above that the new proposed method requires more resources to learn from data and form a FL model. In this Section, we perform analysis of the time required for learning and the space to be occupied. The first aspect is given with respect to single- and multi-threaded applications that are used in modern computational systems.

Computational Complexity of the Proposed Method

The estimation of the computational complexity is an important issue to be considered in critical applications of real-time systems and data analysis. We measured execution time for the training and rules inference parts of the *mobile malware* dataset as it is shown in the Table 4.47. Using the size from the proposed method, SOM trains faster due to the lower amount of rules. We used the difference between the output's values y_i of two adjacent steps on the 2^{nd} stage of NF as a stopping criteria.

Table 4.47: Time in seconds, required to learn the NF mode using different estimations for optimal SOM and methods for fuzzy patches construction on *mobile malware* dataset with parallel optimization

Method	**Training**, seconds			**Inference**, 10^{-6} seconds		
	Simple	**Kosko**	**Gaussian**	**Simple**	**Kosko**	**Gaussian**
"Rule of thumb" (49 rules)	0.3820	0.5185	0.3402	2.97	3.33	120.00
Vesanto (28 rules)	**0.3805**	0.4715	0.3401	2.53	2.98	101.00
Proposed (24 rules)	0.3833	**0.3647**	**0.1662**	**1.71**	**2.18**	**55.30**

The size of SOM is decided first based on the statistical properties of the data set. In most cases however, the bigger SOM size causes degradation of the accuracy and execution time with a decrease in interoperability. Furthermore, a larger SOM will results in a longer NF training time and a slower fuzzy rules inference process

when there is a need to classify a new unlabelled sample. The proposed method of SOM size determination shows better performance on the training phase as well as the inference phase than the other two methods. This can be explained by well-fitted fuzzy patches and lack of need for longer training on the 2^{nd} stage of NF. Inference by Gaussian MF takes more time however, so a more advanced optimization of the inference might be needed, yet this is out of the scope of this paper. One can see that simple fuzzy patches take almost the same time for all three methods of SOM size determination due to simplicity of used triangular MF.

Parallel Optimization for Growing Number of Fuzzy Patches

In the Table 4.47, we present the time required to learn and to make decisions for simple, Kosko, and Gaussian MF. The time measurements are given for single-threaded and multi-threaded applications. For testing purposes, we took the *Mobile malware* dataset whose results are presented in the Table 4.48 for SOM sizes ranging from 9 to 100 nodes. It has been measured by the execution of the implemented version in a single- and multi-threaded mode.

Table 4.48: Time required to learn three types of NF models with respect to three methods of SOM size determination using 6 parallel threads

SOM size	Extr. rules	MF Method	**Sequential** , sec	**Parallel** , sec
3x3	16	Simple	0.3958	0.3619
		Kosko	0.3936	0.3131
		Gaussian	**0.3825**	**0.1239**
5x5	35	Simple	**0.5534**	0.3912
		Kosko	0.8001	0.4723
		Gaussian	0.8237	**0.2492**
10x10	49	Simple	**0.6415**	0.4771
		Kosko	1.1222	0.6571
		Gaussian	1.1539	**0.3567**

The computational complexity is an important issues that affects decision whether to use particular methods while dealing with data complexity that will affect the execution time. The proposed method of SOM size determination requires less time than *"Rule of thumb"* and *Vesanto* methods when *Parallel execution* is used. As we can see from the Table 4.45, the accuracy of the proposed method is much better. Also, the utilization in systems with a bigger amount of CPUs (tens of execution threads) may yield even better results. However, the simple MF model uses less time when it comes to *Sequential execution* due to simplicity of the MF and corresponding calculations.

Space Complexity

In order to estimate the required space (memory) complexity of the Kosko and proposed scheme for storing the fuzzy rules parameters, we modelled both stages of NF using the same clustering results from SOM. In this experiment, we did not estimate the size of the Rectangular fuzzy patches since its and Kosko fuzzy patches use triangular MF and occupy the same amount of space. The results of the experiment for the *Mobile malware* dataset model are presented in the Table 4.49. The dataset has 41 features and 596 data samples, which results in 24 rules using the proposed method of SOM size determination, 39 for *Vesanto* and 48 for *"Rule of thumb"* methods. The *structure* describes the necessary amount of memory to initialize storage for each rule without data inside, *rule* shows the memory needed for each rule, while *model* presents total amount of memory including storage overhead required for the extracted classification fuzzy model.

Table 4.49: Comparison of the size fuzzy rules for two types of MF using different architectures: 32 and 64 bits. The measurements are: Structure - size of empty rule structure, Rule - size required to store a single rule, and Model - total size required to store all the classification rules.

SOM size	**Architecture**	**MF types**	**Required size & overhead**, Bytes		
			Structure	**Rule**	**Model**
"Rule of thumb" 48r.	32 bit	Triangular	24	872	41,856
		Gaussian	28	15,848	760,704
	64 bit	Triangular	48	1736	83,328
		Gaussian	56	31,688	**1,521,024**
Vesanto 39 r.	32 bit	Triangular	24	872	34,008
		Gaussian	28	15,848	618,072
	64 bit	Triangular	48	1,736	67,704
		Gaussian	56	31,688	**1,235,832**
Proposed 24r.	32 bit	Triangular	28	872	19,184
		Gaussian	24	15,848	348,656
	64 bit	Triangular	48	1,736	38,192
		Gaussian	56	31,688	**697,136**

The proposed radial-basis rules need to store $M \cdot M$ elements of the inversed covariance matrix + N_S centroids, while triangular needs only the $2 \cdot N_S$ centroids. This means that the size of the rules in data analysis will converge to a number of samples, which means that if $M << N_S$, then the size of the elliptic Kosko and Rectangular rules will exceed the size of the Gaussian MF making them more demanding in storage complexity. The size of the proposed model can be saved by exploiting the symmetric property of the covariance matrix. Otherwise, the proposed rules will occupy more space on the small sample. As can be seen from the Table 4.49, the complete model of 24 rules can be stored in RAM using ≈ 38

KBytes for triangular MF rules and $\approx$ 750 KBytes for the proposed MF on 64 bit PC. The *Vesanto* and *"Rule of thumb"* methods require at least 2 times more space for the model, exceeding 1MByte. The size looks reasonable considering the capacities of modern computers. This makes it possible to apply the rules on the embedded devices with a significant resource limitation, since the vectors are stored in contingency memory and don't require multiple random accesses for inference. Modern mobile devices provide 0.5-1GB of RAM with much greater storage space, which is sufficient for use by the proposed model.

Summary: This subsection addresses the problem of optimal SOM size determination used on the 1^{st} stage of Neuro-Fuzzy. It was studied how the trade-off between the accuracy of the resulting NF model and the interpretability can be achieved with respect to the malware analysis in Digital Forensics. We have proposed a new, efficient, and fast approach to deriving an optimal number of nodes in SOM to be clustered using data analytics rather than an alteration of the dataset. Experiments on different datasets proved the applicability of the method using simple triangular, Kosko, and Gaussian MF with corresponding rectangular and elliptic fuzzy patches configuration. The proposed method decreases the number of effective rules from 42 in *Vesanto* and 39 in "Rule of thumb" methods to 24 in the case of Android malware detection, based on versatile features collected from static and dynamic tests. The number of suggested rules is decreased from 189 in the Vesanto method to 13 in proposed method, and the classification accuracy grows due to a smaller spread of the clusters, achieving 92%. Finally, the proposed method uses less time to train a fuzzy model compared others while applying Gaussian MF.

4.6.6 Dynamic Feature-based Expansion of Fuzzy Sets in Neuro-Fuzzy for Proactive Malware Detection

In this subsection, we present the results of the proof-of-concept demonstration of the proposed DENF method for malware analysis.

The dataset was collected from real Android OS applications, so it might contain different means of functionality obfuscation, encryption, etc. Later, the following features were extracted from the original dataset using the Information Gain feature selection method: *targetSdkVersion, permissions_highest, permissions_avg, package_number_files, and manifest_size, classes_dex_size* . These features were transformed into the fuzzy sets with a defined number of fuzzy terms. Note that the final dataset includes 6 features out of 36 that are proven to be efficient in malware detection with a high classification rate under cross-validation by SC and other methods. Furthermore, we define several cases that expose the implications of the proposed improvements in Hybrid NF re-training over the conventional method.

For the experiments, we created general fuzzy sets of each of the aforementioned numerical features and split them in two commonly used sets of terms. Gaussian membership functions with corresponding means and spread were used, since according to the three-σ law, such an approach provides the coverage of the large part of the distribution of the numerical feature. Moreover, this offers a naive approach for automated fuzzification.

For the 3-terms set, we used following terms:

$$\begin{aligned}(LOW, MEDIUM, HIGH) \; &= \\ &(\mu - 2 \cdot \sigma; \mu; \mu + 2 \cdot \sigma)\end{aligned} \tag{4.18}$$

For 5-terms set we used the next terms:

$$\begin{aligned}(VERY\ LOW, LOW, MEDIUM, HIGH, VERY\ HIGH) \\ = \; (\mu - 3 \cdot \sigma; \mu - \sigma; \mu; \mu + \sigma; \mu + 3 \cdot \sigma)\end{aligned} \tag{4.19}$$

Experimental Design

In fact, we created the environment that simulates the learning of Hybrid Neuro-Fuzzy prior to and after adding the new term. To evaluate the performance of the method, we used an accuracy measure:

$$Acc = \frac{N_{benign} + N_{malicious}}{N} \tag{4.20}$$

where N_{benign} is the number of properly classified benign samples, $N_{malicious}$ is the number of properly classified malicious samples, and N is the total number of software samples.

The Fuzzy Logic part of NF used automatic statistical parameters allocation for each of the 3 fuzzy sets. The ANN part of NF was initialized with initial weights values 0.5, a fixed learning rate of α 0.1, 100 training epochs, and the first 10 rules were selected based on importance[11], which will be mentioned later in the justification for these parameters. On this dataset, the one-layer ANN provides classification accuracy of 86.9205%, which will be the reference point for our further comparison.

Results & Analysis

Finally, we simulated the following cases that shuld be considered while constructing pro-active malware or intrusion detection systems:

[11]The importance of the selected rules was taken as proportional to the rules weights. The logic behind this was that the higher weight will result in voting for the rule, which denotes its importance.

Case 1 - Fixed set of terms in a fuzzy set. The NF was trained using the 3 linguistic terms in each fuzzy set. The results are shown in the Table 4.50. This case refers to an initial learning of the NF based on the given data set.

Case 2 - A single linguistic term is added to a single fuzzy set variable. A new term *VERY HIGH* is being added to the fuzzy set *targetSdkVersion* besides the existing 3 terms in this set. Then, the comparison of the performance for NF[12] and DENF was made. The results are presented in the Table 4.50.

Method	**Case 1**, %	**Case 2**, %	$T^{train}_{par.}$, sec	$T^{train}_{seq.}$, sec	$T^{slct.}_{par.}$, ms	$T^{slct.}_{seq.}$, ms
Simple NF	83.61	82.62	12.1351	99.3929	2.1	17.0
DENF	-	**82.95**	2.9108	29.4822	**1.0**	1.0

Table 4.50: Accuracy, required re-training, and rules selection time with and without parallel optimization

The DENF is not used for Case 1 since the given dataset is fixed and only general NF was trained from it. There is an increase in the accuracy from using DENF for the set that can be explained by the concept drift and the outdated information in the trained NF. The time required for re-training of the DENF against training of NF from scratch is *12.1351* seconds against *2.9108* seconds respectively that is considerably significant in terms.

In addition to this, we have cross-validated the results of our method's accuracy using community-accepted Machine Learning methods implemented in Weka, as shown in the Table 4.51. SVM has the lowest possible accuracy of 66.8874%, which is on the lower end for binary classification problems. Further, C4.5 shows the accuracy of 91.3907% with 29 leaves and the size of the tree equal to 57. The highest accuracy of 93.2119% was achieved using k-NN. Even though we did not find that the method can support dynamic fusion of the rules, the proposed method is at least consistent with the presented results.

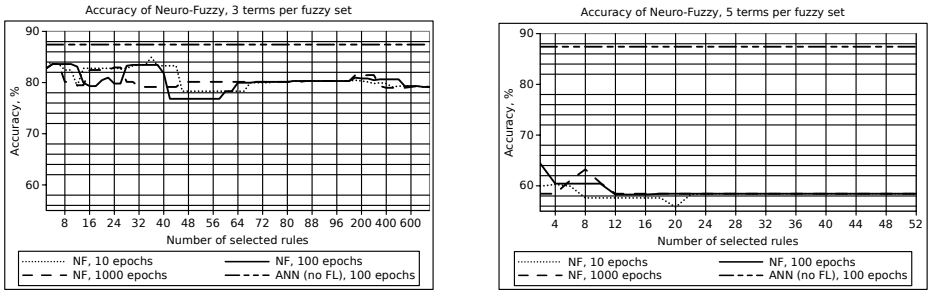
Method	SVM	ANN_{1layer}	$\text{MLP}_{20layers}$	K-NN	NaiveBay.	JRip	C4.5	BayesNet
Accuracy	66.88	87.08	92.21	93.21	87.25	91.72	91.39	91.05

Table 4.51: Accuracy of ML methods on the dataset, in %

Dependence of Accuracy on Number of Fuzzy Rules

During the experiment design, we performed a study of the performance of generic Neuro-Fuzzy with respect to the number of terms in fuzzy sets as it is shown in

[12]Note that the time to retrain NF is be the same as initial training



(a) **(b)**

Figure 4.29: Accuracy of Neuro-Fuzzy model using 10,100, 1000 epochs in ANN training with selected number of fuzzy rules $NS \leq NC$ and reference ANN accuracy

the Figure 4.29. The figure shows that the smaller a fuzzy set is, the more robust are its results in classification. Additionally, the increase in the amount of training epochs may give better results, though the training time depends linearly on the amount of epochs.

The amount of rules should be reduced to keep the model simple and understandable. Therefore, one of the main challenges to rule selection is the selection criteria and amount limitation. A smaller amount of rules brings more certainty in malware detection while a model without pruning possesses a higher error rate. Thus, we proposed a selection method based on the significance of the rules using weights of the trained model. It was also observed that in the proposed method, the values of the membership function of the rules with less importance are negligibly low, and can therefore be eliminated.

Summary: The main idea of this research is to show how pro-active malware analysis can be optimized with respect to the need for dynamic changes in the model by means of adding new software characteristics without complete model re-training. Since the Neuro-Fuzzy approach does not possess the flexibility in the re-training of the model when the fuzzy sets are changed, we have proposed a DENF method. It is based on the fuzzy rules γ-memory from the initial training and δ-memory that contains newly contracted fuzzy rules based on the added characteristics. Also, we used the automated allocation of the statistical parameters in each fuzzy set based on the analysis of the given dataset. To demonstrate its efficiency, we considered the performance of the DENF and NF methods i two cases: in fixed fuzzy sets and when a single new linguistic term in a fuzzy set has been added. The time needed to execute DENF is in nearly 4 times less than is needed for NF to re-train, while the accuracy is increased from 82.62% to 82.95%

when a new term has been added in the fuzzy set. Pointedly, the accuracy is less than in ANN since an abstraction level is added. Also, Fuzzy Logic must be tuned manually, and in this case the automated extraction of fuzzy rules parameters was used. For future work, we foresee research on the deletion of the terms not only in a single fuzzy set, but also across multiple sets. Additionally, model fusion can be proposed for more efficiency in dealing with both initial and re-trained models parameters. Finally, the lightweight frameworks can be constructed for use as Decision Support System.

4.7 Privacy Preserving & Access Control[13]

Artificial Neural Network (ANN) is one of the most powerful methods capable of modelling complex non-linear relations between input data and the output decision. It has been successfully applied in a number of fields in Information Security, such as Intrusion Detection [315] and web attacks classification [372]. Access Control (AC) is one of the most important tasks of Information Security. Modern AC models and policies are based on a specifically predefined set of rules for a single user or group of users. The challenge comes when dealing with agile environments like grid-systems or a cloud environment according to Bedi et al. [74], making traditional access methods less efficient. Below we target the on-line learning of the AC mechanism based on the flow of access logs data. Despite the wide applicability of Multilayer Perceptron (MLP), which is a type of ANN, its usage in data streams mining brings additional limitations such as the availability of data in a very short time period. MLP is capable of providing fast responses [350] however, so we believe that the optimization of MLP learning may facilitate similarity-based AC.

4.7.1 Dataset

The Kaggle Amazon Employee Access Challenge [211] was used as an access log to test the proposed method. It consists of training 32,768 data records characterized by a 9 integer-valued feature as access requests: ACTION, RESOURCE, MGR_ID, ROLE_ROLLUP_1, ROLE_ROLLUP_2, ROLE_DEPTNAME, ROLE_TITLE, ROLE_FAMILY_DESC, ROLE_FAMILY, ROLE_CODE. ACTION is a binary class label: 0 - action against resource is denied (1,897), 1 - action is approved (30,871). Since the original labeled testing dataset of the challenge is not published, we decided to use the model by Duan et al. [129] with the performance of AUC = 0.92360 as a reference testing dataset that consists of 58,921 test data samples.

[13]The main ideas of this section are published under the contribution [370]

4.7.2 Experimental Design

For proof of concept demonstration, we used only 3 out of 6 layers according to the "Rule of thumb" optimal $N_{hidden} = \frac{2}{3} \cdot (N_{In}+N_{out})$, and defined a generally accepted fixed learning rate $\alpha = 0.3$. Also, the amount of epochs for off-line learning was defined as 10 and 1 for the optimized single-step model.

These metrics provide an opportunity to compare the results of an optimized MLP model with other implementations. The implementation is capable of scaling and expanding to any number of features and hidden layers. The MLP architecture used has a rectangular unbiased structure, since the idea of the model was to find an appropriate set of learning rates over a single step. That means that as a single-control point of the layer, bias might be irrelevant because it requires manual adjustments [232]. The Figure 4.30 shows the experimental design using the proposed method.

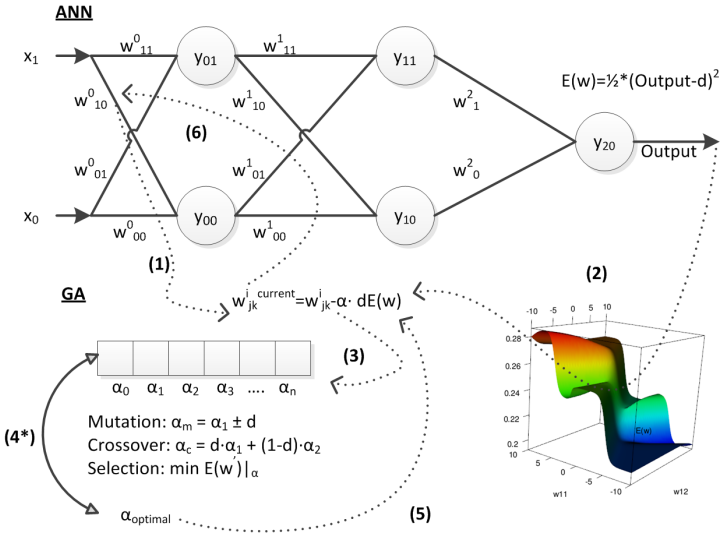


Figure 4.30: Proposed method for single-step on-line learning

We assumed that a set of pre-defined access policies and known access patterns could be found by selecting a specific fraction of the data for training before the data stream is fed to the model. The following experiments were conducted: (i) 100 samples were used for training, followed by the data stream of test data. (ii) 1,000 samples are used as training data and then testing is done for the new samples from the data stream. We also performed a standard batch learning and cross-validated the accuracy of the implemented method in C++, in comparison with community-accepted implementations in Weka [149] and RapidMiner [14]

as shown in the Table 4.52. The optimized model used 10 iterations of GA with 10 chromosomes in each population, and 4 mutation and 7 crossover operations; other MLPs used 100 epochs. The results look consistent. Generally, RRSE is a description of the mean prediction. In this case, we can see that the mean is biased towards an approved action "1", as the majority of access patterns in a system are legitimate. Therefore, all implementations show a similarly high value of RRSE.

Method	**MAE**	**RMSE**	**RRSE**, %
MLP impl.	0.061	**0.140**	**100.849**
MLP impl. + GA	**0.054**	0.142	102.665
Weka MLP	0.061	0.149	107.554
RapidMiner MLP	0.059	0.151	108.700

Table 4.52: Performance of implementations on static dataset

We performed experiments having a data stream with a limited availability of data samples for training.

4.7.3 Performance Evaluation

For the comparison of the implemented model on the testing data, several metrics were used such as Mean Absolute Error (MAE), Root Relative Squared Error (RRSE), and Root Mean Square Error (RMSE), considering that the data streams mining model is regressional according to Bifet [80].

$$\begin{aligned} MAE &= \frac{1}{N} \cdot \sum_{i=0}^{N-1} |y_i - d_i| \\ RRSE &= \sqrt{\frac{\sum_{i=0}^{N-1} (y_i - d_i)^2}{\sum_{i=0}^{N-1} (y_i - \bar{d})^2}} \\ RMSE &= \sqrt{\frac{1}{N} \cdot \sum_{i=0}^{N-1} (y_i - d_i)^2} \end{aligned} \tag{4.21}$$

4.7.4 Results & Analysis

Similarity-based Access Control performance

We performed single-step on-line learning of the MLP as well as cross-validation of incoming data in the data stream S. The experiment was built as follows: the boundaries of α for GSS and GA were chosen to be $[0, 1]$. The MLP started from randomly initialized weights and trained with $l\%$ samples from the training dataset. The stream of $k\%$ samples from the classified test dataset is then fed to the

model. Moreover, the trained dataset constantly trains the model while new test data samples arrive. Finally, the earlier-defined performance metrics are computed over classified samples from the test dataset. The Table 4.53 represents the results for the on-line scenario. The GSS method used the same number of iterations as number of epochs in the GA. Original MLP without optimization trained 10 epochs, and MLP with optimization only one.

Table 4.53: Performance comparison of MLP on test dataset in on-line incremental learning using optimized and non-optimized techniques in data stream scenario

Method	**MAE**	**RMSE**	**RRSE**, %
start with 100 pre-training samples			
MLP (1 epoch)	0.065	0.180	36.117
MLP (10 epochs)	0.070	0.179	35.933
MLP + GSS (1 epoch)	0.060	0.173	34.682
Proposed	**0.054**	**0.167**	**33.567**
start with 1000 pre-training samples			
MLP (1 epoch)	0.056	0.155	32.184
MLP (10 epochs)	0.057	0.155	32.183
MLP + GSS (1 epoch)	0.052	0.150	31.210
Proposed	**0.038**	**0.140**	**29.078**

The results show that the proposed method gives better accuracy on all performance metrics for 100 and 1,000 access log samples that were available for the initial pre-training. However, such an application of GA for on-line MLP optimization has higher computational complexity cost than the simple utilization of constant α. Therefore, we believe that parallel optimization can help to achieve fast processing speed, since sequential execution would be much slower than standard MLP. This can be done using modern CPU and GPU.

Error surface influence on MLP training

The fitness function used in GA is basically the same as the error function in MLP $E(W)$. The derivative on each step is estimated in the neighbourhood of the corresponding neuron weight w with precision $h = \pm 10^{-6}$ in error cost function $E(W)$, while keeping constant all other weights and the input data sample. To study the dependency between the weights change and error function, two arbitrary weights were chosen on the lower layers (w_3^1 and w_9^1) in the 3-layers perceptron. Figure 4.31 shows the surface of the error function for two arbitrarily-selected weights. Unless some sophisticated learning rate updating methods and stopping

criteria are used, the EBP will converge to a sub-optimal solution.

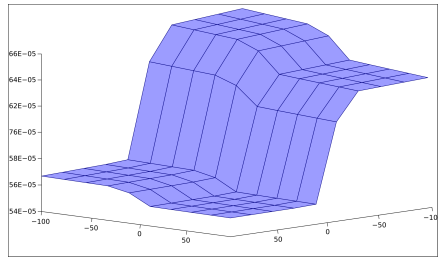
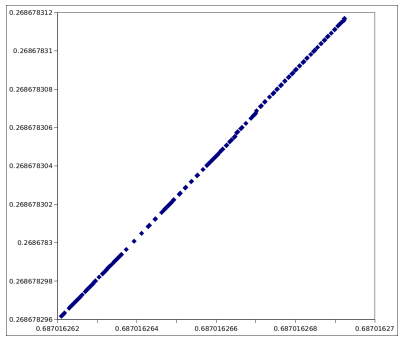
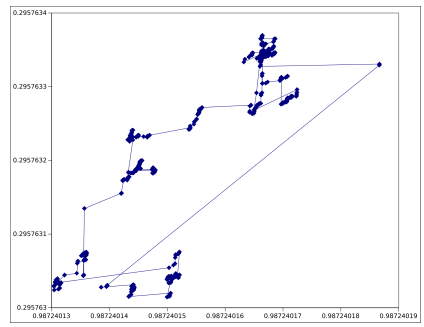


Figure 4.31: Surface of the error function showing dependency of $E(W)$ on w_3^1 and w_9^1 as covariates in 3-layers MLP that was trained from the given dataset

The Figure 4.32, (a) represents an example of the path of both weights along 1,000 iterations until it gets converged. The learning rate stayed constant, showing that the path is gradually following one direction. On the other hand, Figure 4.32, (b) shows that the proposed method has broader coverage of the search space, giving a higher chance of achieving an optimal solution for the non-linear error function.



(a) Conventional MLP training with a constant learning rate α



(b) Proposed method for individual learning rate α optimization

Figure 4.32: Path traverse of the weights w_3^1 and w_9^1 in MLP

Considering smooth transition between minima and the neighbourhood of the error function in the Figure 4.31, we can see that the proposed method will likely result in a better solution than when training ML with a constant step. Therefore, MLP can be trained using adoptive rate α approach with a higher number of layers faster rather than with a fixed-rate iterative line search.

Summary: Conventional off-line learning of Multilayer Perceptron is no more re-

liable for data streams mining since it requires more iterations to learn and does not converge quickly. As a result, it cannot be successfully applied for on-line training of Similarity-Based Access Control models, where the attributes of resource and user can be utilized to evaluate whether similar users may access similar resources. To overcome this limitation, a single-step learning of Multilayer Perceptron can adjust the model once new access requests come, and does not require constant access to that pattern. In this thesis, the Evolutionary Computing method was utilized, Genetic Algorithm in particular. We consider this method to be more robust in an environment with a consistent drift of the data statistics and non-deterministic events. The influence of the weights of the layers on the error function was also studied, and we can state that it requires better optimization due to non-linearity. Therefore, we proposed how the appropriate learning rate α calculation can be done using Genetic Algorithm for each neuron in accordance with an optimal solution on each layer. This eliminates the need for constant iterative batch learning of the Multilayer Perceptron, allowing the model to be a single step. Finally, one of the main benefits of the proposed method is that it scales and can be used for larger non-linear Neural Networks.

Chapter 5

Summary & Future Work

In this chapter, we provide a brief overview of achieved results and discuss their implications for future work. Big Data is a reality and Cyber Crime Investigators are confronted with a great amount and complexity of seized digital data in criminal cases. Human experts are sitting in Courts of Law and making decisions with respect to evidence found and presented. Therefore, there is a strong need to bridge data processing and automated analysis for providing representations of human-understandable results. There is a history of the successful application of Machine Learning methods in Digital Forensics, such as Artificial Neural Networks, Support Vector Machines, and Bayes Network. The challenge with this approach, however, is that such methods neither provide human-explainable models nor can work without the prior knowledge required for inference and data representation. In this thesis, we focus on Neuro-Fuzzy, a Hybrid Intelligence method that is capable of connecting two worlds: Computational Intelligence and Digital Forensics. Generic Neuro-Fuzzy methods showed poor performance on a class of Digital Forensics data analysis problems and required specific improvements in order to be applicable. So, our goal was to revise this method to be able to extract fuzzy rules that would be further presented in a Court of Law, rather than a set of weights or parameters generated by other models. Also, the enhancement of Neuro-Fuzzy by Deep Learning shows that the results of a data-driven approach can be fine-tuned for modeling a high-level abstraction in complex data.

5.1 Summary of Findings

This thesis is inspired by the exponential growth of data stored on digital carriers that are being seized in Cyber Crime Investigations. In most cases, the manual work of a forensic expert is required to process a large amount of data and extract

a human-understandable model. The challenge with this lies in the amount and complexity of data. Below, we summarize our work on automated rule-extraction and classification for Digital Forensics. In this Section, we present the general findings that were made during work on the dissertation.

5.1.1 Main Contributions

The contributions are divided into the following sub-areas: (i) Study of applicability of Machine Learning and Soft Computing methods in particular for Digital Forensics, (ii) Multi-part improvements of Neuro-Fuzzy upon a comprehensive analysis of the method's drawbacks and requirements in advanced data analytics, (iii) Multinomial classification by Neuro-Fuzzy as a non-trivial problem, and (iv) a novel large-scale dataset of PE32 files that includes multiple families and categories.

Soft Computing in Digital Forensics

One of the contributions of this thesis is a survey and description of static-based malware detection using Machine Learning techniques. After a preliminary literature study, we performed an extensive review of the relevant scientific works. We provided an in-depth overview of static malware characteristics with corresponding feature construction methods. Furthermore, we created a taxonomy based on a comparison of different cases of ML methods, which will be given with guidelines for the usage and utilization of available implementations. Later, we gave a tutorial on how these methods are applied in real large-scale scenarios. We also discussed the applicability of each particular method based on its achieved accuracy. Thus, this thesis contributes as a survey of the existing state of the art in malware analysis using machine learning. We believe that static analysis has great potential, and can facilitate large-scale malware detection; this is the answer to the RQ 1.

Additionally, we studied Soft Computing as a sub-field of Machine Learning. Given Digital Forensics requirements for large-scale machine-aided analytics, Soft Computing methods are prominent among Computational Forensics approaches. There is a high demand for the automated analysis of large-scale data in different fields of Forensics Science, considering the emergence of Big Data. In this thesis, we presented an insight into how Soft Computing can be used to provide not only explainable, but also a forensically sound analysis of evidence data. Furthermore, it is possible to combine SC with search models in order to obtain explainable Computational Forensics models. Moreover, we looked at the optimization by meta-heuristic methods that improve the required performance when dealing with Big Data challenges. We also presented an overview of our own work done to improve the performance of Soft Computing in Forensics Science by means of

hybridization. So, to answer the RQ 2, Neuro-Fuzzy methods and its improvements approached the fields of Forensics Science such as malware analysis and network forensics. Also different aspects and improvements of information fusion and on-line learning were considered as a part of answer on the RQ 3.

Improvement of Neuro-Fuzzy rule-extraction classification model

As was found, Neuro-Fuzzy can provide the accurate and human-understandable data representation required by Digital Forensics testimony in a Court of Law. However, the fuzzy inference system proposed by Kosko [233] based on Mamdani-type linguistic classification rules is not suitable due to its intrinsic inability to handle complex and large-scale data. To handle this, an improved NF was proposed to facilitate large-scale analysis in digital forensics. A number of applications were successfully studied. We made a synergy of optimal SOM parameters inference, the most appropriate method of fuzzy regions construction and corresponding MF that can be derived to incorporate all possible information on the 2^{nd} step of NF. The new method showed great improvement in performance and the required learning and inference time in comparison to classical approaches. Additionally, it was observed that bootstrapping on the 1^{st} stage of NF may result in a lower amount of rules and a more generalized classification model based on the fuzzy rules. The proposed theory is mainly an answer to the RQ 3 and RQ 4.

A novel Portable Executable 32bit dataset

One of the main objectives of this thesis was a feasibility study of the application of Neuro-Fuzzy for large-scale data analytics problems that exist in the Digital Forensics area. There already exists the KDD CUP 1999 dataset for Network Intrusion Detection, even though it is considered outdated and old when dealing with firewalls and IDS testing. However, such a public dataset is missing in the area of malware analysis. Therefore, we originally collected 400k PE32 malware species for MS Windows OS, later filtered down to 328k malware samples. It can be considered a large-scale dataset suitable for our experiments since the datasets used in studies before are mostly small sets with thousands and tens of thousands of samples [333, 458, 385, 82]. Finally, we investigated large-scale malware detection using Soft Computing methods based on the static features extracted from PE32. It is important not only to distinguish between benign and malware files, but also to understand what kind of malicious file it is: malware family and type. To explore this problem, we created a novel dataset of malware features with respect to families and types using publicly available sources and tools. By building this dataset, our intention was to facilitate an answer to the RQ 5.

Multinomial classification in Neuro-Fuzzy

A majority of work in the area of Computational Intelligence for security normally considers mostly binary classification problems where there are defined "benign" and "malicious" classes. This approach can be found in both malware analysis and intrusion detection. On the other hand, one of the few works that considered aspects of multinominal classification was written by [303] and is devoted to Artificial Neural Networks. So overall, one can see that there is a lack of contribution towards multi-class problems and corresponding Neuro-Fuzzy training.

In this thesis, we proposed a set of improvements towards the application of Neuro-Fuzzy in multinomial classification problems. Digital Forensics requires one to find not only *"benign"* or *"malicious"* activity patterns, but also to distinguish between multiple *"malicious"* patterns. However, a conventional single-output NN method must work with either two sets of classes or alternatively requires additional outputs per class. In order to overcome multiple outputs encoding in the Neuro-Fuzzy, we suggested using a single-output mode for reduced overhead and training time. The corresponding Center of Gravity defuzzifier was modified to be compliant with Mandani-type rules as well as to incorporate class labels. Finally, we established a baseline for creating Deep Neuro-Fuzzy that has great potential when combining human-perceivable computational models and higher-level abstractions. This part provides answers and considerations to the RQ 5 and RQ6.

5.1.2 Overview of Main Results

During our thorough analysis of the Neuro-Fuzzy rule-extraction classification method, we found that its application for Digital Forensics is promising, but holds a number of drawbacks such as (i) an inability to learn from real-world data in comparison to state of the art methods, (ii) the output model is often too large, so human experts will not be capable of understanding it, (iii) a strong overfitting in case of large-scale data caused by a huge number of rules, and (iv) part of the training data is neglected since the intrinsic learning procedure does not represent it in the final classification model. Because of this, Neuro-Fuzzy has been under criticism and has not been widely used in the area before. Our research has targeted the improvement of Neuro-Fuzzy and facilitated its application in Digital Forensics.

Self-Organizing Feature Map Parameters Inference

We have proposed a new method for forensically-sound fuzzy rules construction using NF on the optimal SOM size determination using exploratory data analysis. The main idea was to produce an accurate model with a moderate amount of fuzzy

rules that will also be human-understandable, and that can be used in Network Forensics Readiness effectively. The known Vesanto method for SOM size produces a vast amount of clusters that result in complex models that require enormous computational resources when dealing with Big Data in Digital Forensics Investigations, particularly with large-scale datasets that contain millions of data samples. Better generalization of the Neuro-Fuzzy method using the exploratory analysis of data for SOM clustering was proposed, particularly targeting large-scale datasets.

The proposed improvements of the 1^{st} NF stage along with bootstrap aggregation gives considerably better performance both in terms of accuracy and required training time. We have studied the influence of bootstrap aggregation on the final model using 1% of the dataset. We can state that it can be used to improve the generalization of the extracted clusters, though the accuracy might degrade. Also, the Vesanto method is not suitable for use with bootstrap aggregation due to the lower number of samples available for clustering.

Elliptic Fuzzy Patches Estimation

Big Data analytics requires new and enhanced models to handle complex problems such as Network Forensics Readiness and network traffic analysis. The conventional Neuro-Fuzzy method is very much affected by overfitting, reduced explainability of the classification model, and enormous training time when it comes to million-sample sized datasets. As a result, faster and more accurate models are required. Therefore, the proposed improvements can serve as a stepping stone for real-time protection and forensically-sound evidence collection in network environments.

Kosko [233] proposed a better way of data description than simple rectangular patches. It used elliptic fuzzy patches for function approximation with an empirically-estimated pseudo-radius α, which must be derived experimentally. In contrast, the new method for fuzzy patches construction on the 2^{nd} NF stage improves classification accuracy by automating the estimation procedure. We can see that a lower number of fuzzy rules requires more advanced techniques to incorporate information from data than using the Kosko method with triangular MF based on the projections. The proposed method for fuzzy patch construction uses χ^2 tests to find the parameters of each patch. The method was implemented and showed better accuracy results and learning time on a desktop computer using four different datasets, each having an average amount of over 5 million samples. Current Machine Learning methods and their implementations are designed to handle tens of thousands of data, yet have complexity issues with bigger sets. Thus, the estimation of the pseudo-radius α of a hyper-ellipsoid can be done automatically using a

χ^2 goodness of fit test rather than a manual approach in the Kosko method.

Gaussian Membership Function

Simple triangular membership functions show high error levels when learning complex high-dimensional data. Kosko [233] developed a better triangular membership function based on the projections of hyperellipsoids on features axes. However, the main drawback is that it does not incorporate data distribution and correlation properly. To mitigate this, Gaussian approximation of the data distribution offers an improved data approximation and better Neuro-Fuzzy model accuracy than classical rectangular fuzzy patches. A Modified Gaussian Fuzzy Membership Function provides a robust estimation of membership degree with respect to data properties, which can't be achieved with triangular or Kosko projection-based Membership Functions. The achieved accuracy of the proposed improvement was 98.787% when using 39 rules against 98.790% with 10,231 rules when applying the original method. For future research, we will investigate the utility of non-parametric models for elliptic regions estimation.

Multinomial Classification

Improvements were proposed towards the application of Neuro-Fuzzy in multinomial classification problems, such as web attacks detection. A majority of Digital Forensics applications require finding not only *"benign"* or *"malicious"* activity patterns, yet also distinguishing between multiple *"malicious"* patterns. However, a conventional single-output Neural Network-based method must work either with two sets of classes or alternatively requires additional outputs per class. In order to overcome multiple outputs encoding in the NF, we proposed to bound clustering results of SOM for better statistically-sound fuzzy rules parameters as well as apply a modified Gaussian membership function. Then, we suggested using a single-output mode for reduced overhead and training time. The corresponding Center of Gravity defuzzifier was modified to be compliant with Mandani-type rules as well as to incorporate class labels. Reconsideration of the application of Neuro-Fuzzy for multinomial problems yields positive results, showing the need for more advanced data modeling. Later on, a new approach to modeling non-linear data using the standard Neuro-Fuzzy method results in using Deep Learning. The deep Neuro-Fuzzy approach is designed to handle highly non-linear data while giving robust classification, and outperforms contemporary Neuro-Fuzzy which has considerably lower accuracy on multinomial problems.

A novel dataset for Windows malware analysis

A novel large-scale collection of the Portable Executable 32bit malware files was composed as a part of this PhD thesis. It consists of 328k labelled malware families

(10,362) and categories (35), which was further tested as a non-trivial multinomial classification problem, neither sufficiently studied in the literature nor explored before.

5.2 General Considerations

Below, the outcomes of this dissertation will be presented along with a critical analysis of the achieved results and their possible application.

5.2.1 Theoretical Implications

In this thesis, we focus on improving the Hybrid Intelligence method Neuro-Fuzzy, a combination of Artificial Neural Networks and Fuzzy Logic. As was mentioned in the beginning of the thesis, the biggest challenge in building a fuzzy rules model is seeking for an optimal trade-off between accuracy and interpretability. In the research [199], Ishibuchi stated it as a multi-objective problem as follows for the fuzzy-based system F:

$$\max_{S} \left(Accuracy(F),\ Interpretability(F)\right) \tag{5.1}$$

It is not easy however for a forensics analyst to define the appropriate parameters, especially when dealing with Big Data. The accuracy can be measured easily, while the interpretability measures are not easy to define. Some of the main features used to study their interpretability were number of rules, number of MF, etc. The number of conditions in the rules should not exceed 7 ± 2, which is a boundary that human brain can handle according to Gacto et al. [156]. However, the number of features sometimes can be much more than this limit. Thus, the only way to limit the complexity of the model is to shorten the number of rules. According to Alonso [53], the maximum number of rules acceptable by the user should be greater than or equal to that 10^3 times number of classes. Therefore, as has been shown, the pre-stage that is based in exploratory data analysis simplifies the resulting fuzzy rules-based classification model.

Because of the promising utility for Digital Forensics, we believe that an improved Neuro-Fuzzy rules-extraction classification model can find application in other domains of Information Security where there is a need to build accurate linguistic models for classifying malicious and benign or other activities. Additionally, we studied the performance of binary- and multinomial classification methods and can say that multinomial classification models generally perform better and faster on all sets than ensembled binary classifiers. However, not all of them can produce an understandable model as in the case with NF. Therefore, we believe that it is better to use multinomial NF with the proposed improvements for attacks differentiation to achieve consistent classification results by fuzzy rules. To sum up,

Neuro-Fuzzy is a promising method that is capable of fulfilling the needs of Cyber Crime Investigations towards faster and more accurate Big Data analytics.

5.2.2 Practical Considerations

The amount of digital information being processed in Cyber Crime Investigations is growing exponentially each year. Human experts have been losing their ability to reasonably process such information manually during the last decade. Currently, manual data analytics can be considered an inefficient and expensive measure, and therefore must be focused on decision making rather than a thorough investigation of each piece of data.

Human-understandable Machine Learning model

As there is need to bound the grid size to improve the generalization, it is reasonable to use less than 5^2 rules for the human-understandable model as studied by Ishibuchi et al. [199]. Our experiment was conducted at NTNU Testimon Forensics group during 2014-2016 and includes a comprehensive evaluation of the proposed methods with respect to challenges and requirements in variety of different large-scale real-world applications, such as KDD CUP 1999, PKDD 2007, the novel PE32 files collection, and the SUSY and HIGGS datasets. We discovered that the proposed Neuro-Fuzzy improvements help to considerably decrease the amount of fuzzy rules in the classification model, with a trade-off between accuracy and interpretability.

Large-scale data analysis for Digital Forensics

Due to the growth of network bandwidth, the capacity of digital storage, and the development of new attack scenarios, Digital Forensics faces multiple emerging challenges related to large-scale dataset processing and evidence extraction, as mentioned in the report by Ernst & Young [141]. It is important not only to detect an anomaly and classify it as a likely attack, but also to provide human-understandable Threat Intelligence through a corresponding statistically-based analysis. Conventional Computational Intelligence methods are no longer reliable as they result either in a very complex model that is hard to understand or one that takes too long to infer a meaningful model. Considering the aforementioned obstacles, our contributions to the area through improving Neuro-Fuzzy include mitigation of the following challenges: (1) The large amount of SOM nodes will result in a complex and overfitted model that will be difficult to train when dealing with large-scale datasets. An alternative method for automated determination of the optimal SOM size was suggested. (2) The optimal SOM grid requires more advanced fuzzy patches configuring with the corresponding MF to characterize the data in each cluster better. (3) SOM clustering will most likely produce an

overfitted model due to the large number of specific clusters trained from the data when applied on large-scale datasets. Bootstrap aggregation however improves accuracy and derives a more generalized model. As a result, we proposed the new method to avoid the model overfitting as the Curse of Dimensionality would likely entail, and to reduce the number of extracted rules to as few as possible. We were able to achieve better accuracy using considerably fewer rules not only using the intrusion detection dataset (with five million network traffic packets), but also on other publicly available large-scale datasets. Moreover, our method uses less time and has lower computational complexity in training the classification model and inferring fuzzy rules in comparison to other community-known Machine Learning (ML) methods.

5.2.3 Future Work

During work on this thesis, we identified several directions that can be investigated further, taking into consideration the aforementioned achievements. Big Data is a buzz word that is widely used, but in most cases only means a big number of data records, a single V of the five Big Data Vs. Therefore, the proposed improvements of the Neuro-Fuzzy rule-extraction classification method needs to comply with other Vs such as data Veracity and data Variety. Below, we outline the main ideas for future research.

On-line Learning In Data Streams

As was shown earlier, Neuro-Fuzzy can be successfully used in different fields of Information Security such as network intrusion detection and malware analysis because of its ability to provide a high level of abstraction for complex and incomplete data. Despite its successful application as an off-line learning method, on-line learning can be challenging when dealing with data streams. Data streams mining is a special field that defines such on-line models [446]. From the perspective of Information Security, it includes events monitoring, traffic processing, etc. in addition to access logs analysis [122]. This means that the data are available for a short time frame and should be processed in a fast on-line way rather than an iterative off-line [336] one. On-line learning should be capable of adjusting the parameters of the model from data when a new sample comes. Considering this, we can say that the application of Neuro-Fuzzy in data streams mining can facilitate Information Security, Digital Forensics in particular, and is capable of learning on-line since off-line learning may require significant resources when the organization size is large.

Information Fusion

Information Fusion represents a general methodology used to merge and combine existing data and parameters to achieve better results of more accurate models. Torra [416] stated three possible paradigms that can be used in Information Fusion such as: pre-processing, model building, and information extraction. Our particular interest is in model construction, specifically in combining of several models to achieve a final module simultaneously or over time. This is an inseparable part of the Information Security community, considering the number of sensors that can retrieve information from any computer system.

Taking into consideration that NF application may require dynamic changes in an agile environment, we can state that more advanced methods should be used to update new added terms without complete retraining. It is a particularly critical issue when malware detection requires fast updates of the decision rules while adding new characteristics. The inspiration for this problem came from malware detection using the NF method [371], where the limitations of existing approaches were discovered. For example, when a new feature is added to Android API, there is a need to expand the detection model, and therefore preserve accuracy of the model with and without this feature. Pro-active malware detection needs fast changes in the model without big latency, so NF can be modified with respect to this requirement.

Data Characterization

Most of the data in real-world applications abide normal distribution, also called Gaussian. At least in data used in this thesis, we noticed that the assumption about normality in a variety of different datasets works properly, allowing the proposed improvements to improve accuracy. This means that each measure or property appears around the neighbourhood of so-called "central tendency" with some degree of "spread" or noise. For example, a specific malware family has its own central tendency of measures, such as file size or number of API function calls. This means that unlabeled malware samples that are similar to those measures will likely belong to this family. Guided by this assumption, we utilized χ^2 goodness of fit test for data grouping on the 1^{st} stage of Neuro-Fuzzy. However, future work should consider finding data with other types of distribution, and verifying the performances of such tests on those data in addition to the normally-distributed data.

Bibliography

[1] The 4 V's of Big Data. http://www.ibmbigdatahub.com/infographic/four-vs-big-data. accessed: 12.12.2014.

[2] The R project for statistical computing. https://www.r-project.org/. accessed: 19.04.2017.

[3] Attribute based access control (ABAC) - overview. http://csrc.nist.gov/projects/abac/. accessed: 27.05.2016.

[4] Naming scheme - CARO - Computer Antivirus Research Organization. www.caro.org/naming/scheme.html. accessed: 20.08.2015.

[5] Daubert standard. https://www.law.cornell.edu/wex/daubert_standard. accessed: 20.12.2016.

[6] dlib C++ library. URL http://dlib.net/. accessed: 10.11.2015.

[7] What is soft computing? Techniques used in soft computing. http://www2.cs.uh.edu/~ceick/6367/Soft-Computing.pdf. accessed: 15.01.2017.

[8] Countries with the highest average Internet connection speed as of 4th quarter 2015 (in Mbps). URL http://www.statista.com/statistics/204952/average-internet-connection-speed-by-country/. accessed: 21.04.2016.

[9] History of Internet Explorer. https://en.wikipedia.org/wiki/History_of_Internet_Explorer. accessed: 09.11.2015.

[10] Locards exchange principle. http://www.forensichandbook.com/locards-exchange-principle/.

[11] The malware database. http://malware.wikia.com/wiki/. accessed: 15.07.2016.

[12] PEiD. https://www.aldeid.com/wiki/PEiD. accessed: 2015.11.09.

[13] PeStudio. https://www.winitor.com/. accessed: 25.10.2015.

[14] RapidMiner - 1 open source predictive analytics platform. URL https://rapidminer.com/. accessed: 20.01.2016.

[15] tshark - the wireshark network analyzer 2.0.0. https://www.wireshark.org/docs/man-pages/tshark.html. accessed: 05.07.2016.

[16] UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets.html. accessed: 15.12. 2015.

[17] VirusShare.com. http://virusshare.com/, . accessed: 15.10.2015.

[18] VirusTotal - service that analyzes suspicious files and urls, . accessed: 09.11.2015.

[19] VirusShare BitTorrent client tracker. https://tracker.virusshare.com:7000/. accessed: 18.09.2015.

[20] VX Heaven. http://vxheaven.org/. accessed: 25.10.2015.

[21] Personal data act. https://www.datatilsynet.no/English/Regulations/Personal-Data-Act-/, April 2000. accessed: 07.03.2017.

[22] WinDump. https://www.winpcap.org/windump/, December 2006. accessed: 20.06.2016.

[23] The end of AI winter? http://machineslikeus.com/news/end-ai-winter, October 2007. accessed: 10.12.2014.

[24] CaptureBat. https://www.honeynet.org/project/CaptureBAT, September 2007. 20.05.2016.

[25] ECML/PKDD 2007 discovery challenge - analyzing web traffic. http://www.lirmm.fr/pkdd2007-challenge/, September 2007. accessed: 15.12.2015.

[26] What is soft computing? http://modo.ugr.es/en/soft_computing, 2008. accessed: 04.08.2015.

[27] A survey of access control models. Technical report, NIST, 2009.

[28] Hawking warns AI 'could spell end of human race'. http://phys.org/news/2014-12-hawking-ai-human.html, December 2014. accessed: 13.12.2014.

[29] Attack possibilities by OSI layer. White paper, National Cybersecurity and Communications Integration Center, USA, https://www.us-cert.gov/sites/default/files/publications/DDoS%20Quick%20Guide.pdf, January 2014. accessed: 17.02.2017.

[30] Windows XP support has ended. http://windows.microsoft.com/en-us/windows/end-support-help, April 2014. accessed: 28.10.2015.

[31] 2014-2015 DDoS attack duration and magnitude dataset. Technical report, Akamai Technologies, 2015.

[32] Volatility. http://www.volatilityfoundation.org/, 2015. accessed: 23.05.2016.

[33] Antimalware and antivirus software. http://anti-virus-soft.com/, 2016. accessed: 24.07.2016.

[34] NetMarketShare - desktop operating system market share 2016. https://www.netmarketshare.com/operating-system-market-share.aspx, July 2016. accessed: 13.07.2016.

[35] Stack Overflow - developer survey results. http://stackoverflow.com/research/developer-survey-2016, 2016. accessed: 11.07.2016.

[36] H. Abdi. *Encyclopedia of Social Networks and Mining*. Thousand Oaks (CA), 2007.

[37] L. Abdi and S. Hashemi. To combat multi-class imbalanced problems by means of over-sampling and boosting techniques. *Soft Comput.*, 19(12): 3369–3385, Dec. 2015. ISSN 1432-7643.

[38] A. Abraham. Beyond integrated neuro-fuzzy systems: Reviews, prospects, perspectives and directions. *School of computing and Information Technology, Monash University, Victoria, Australia*, 2002.

[39] A. Abraham. *Adaptation of Fuzzy Inference System Using Neural Learning*, pages 53–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32397-6. doi: 10.1007/11339366_3. URL http://dx.doi.org/10.1007/11339366_3.

[40] A. Abraham. Hybrid intelligent systems: evolving intelligence in hierarchical layers. In *Do Smart Adaptive Systems Exist?*, pages 159–179. Springer, 2005.

[41] A. Abraham and R. Jain. Soft computing models for network intrusion detection systems. In *Classification and clustering for knowledge discovery*, pages 191–207. Springer, 2005.

[42] A. Abraham, C. Grosan, and C. Martin-vide. Evolutionary design of intrusion detection programs. *International Journal of Network Security*, 4: 2007, 2006.

[43] A. Adewuya. *New Methods in Genetic Search with Real-valued Chromosomes*. Massachusetts Institute of Technology, Department of Mechanical Engineering, 1996.

[44] I. R. Adeyemi, S. A. Razak, and N. A. N. Azhan. Identifying critical features for network forensics investigation perspectives. *arXiv preprint arXiv:1210.1645*, 2012.

[45] H. Aguiar, O. Junior, and M. A. S. Machado. Fuzzy firewalls. *Revista Eletrônica de Sistemas de Informação ISSN 1677-3071 doi: 10.5329/RESI*, 5(2), 2006.

[46] K. Ahmad. Fuzzy logic notes. http://www.maths.tcd.ie/~ormondca/notes/Fuzzy%20Logic%20Notes.pdf. accessed:21.09.2014.

[47] S. C. Ahn and A. R. Horenstein. Eigenvalue ratio test for the number of factors. *Econometrica*, 81(3):1203–1227, May 2013.

[48] N. B. Akhuseyinoglu and K. Akhuseyinoglu. AntiWare: An automated Android malware detection tool based on machine learning approach and official market metadata. In *2016 IEEE 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, pages 1–7, Oct 2016. doi: 10.1109/UEMCON.2016.7777867.

[49] A. Al-Mahrouqi, S. Abdalla, and T. Kechadi. Network forensics readiness and security awareness framework. In *International Conference on Embedded Systems in Telecommunications and Instrumentation (ICESTI 2014), Algeria, October 27-29 2014*, 2014.

[50] D. Alahakoon, S. Halgamuge, and B. Srinivasan. A self-growing cluster development approach to data mining. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*. Institute of Electrical & Electronics Engineers (IEEE). doi: 10.1109/icsmc.1998.725103. URL http://dx.doi.org/10.1109/icsmc.1998.725103.

[51] D. Alahakoon, S. Halgamuge, and B. Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *Neural Networks, IEEE Transactions on*, 11(3):601–614, May 2000. ISSN 1045-9227. doi: 10.1109/72.846732.

[52] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *The Journal of Machine Learning Research*, 1:113–141, 2001.

[53] J. M. Alonso, O. Cordón, A. Quirin, and L. Magdalena. Analyzing interpretability of fuzzy rule-based systems by means of fuzzy inference-grams. In *In World Congress on Soft Computing*, 2011.

[54] A. A. Altyeb Altaher and S. Ramadass. Application of adaptive neuro-fuzzy inference system for information secuirty. *Journal of Computer Science*, 8 (6):983–986, 2012.

[55] R. Alvear-Sandoval and A. Figueiras-Vidal. Does diversity improve deep learning? In *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pages 2496–2500, Aug 2015.

[56] M. Aly. Survey on multiclass classification methods. *Neural Netw*, pages 1–9, 2005.

[57] G. Amato. PEframe. https://github.com/guelfoweb/peframe. accessed: 20.10.2015.

[58] F. Amiri, C. Lucas, and N. Yazdani. Anomaly detection using neuro fuzzy system. *World Acad Sci Eng Technol*, 49:889–896, 2009.

[59] E. A. Anaya, M. Nakano-Miyatake, and H. M. P. Meana. Network forensics with neurofuzzy techniques. In *2009 52nd IEEE International Midwest*

Symposium on Circuits and Systems. Institute of Electrical & Electronics Engineers (IEEE), aug 2009. doi: 10.1109/mwscas.2009.5235900. URL http://dx.doi.org/10.1109/mwscas.2009.5235900.

[60] D. Ariu, G. Giacinto, and F. Roli. Machine learning in computer forensics (and the lessons learned from machine learning in computer security). In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, pages 99–104, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1003-1. doi: 10.1145/2046684.2046700. URL http://doi.acm.org/10.1145/2046684.2046700.

[61] J. Armstrong. *Long-range Forecasting: From Crystal Ball to Computer*. A Wiley interscience publication. John Wiley & Sons Canada, Limited, 1978. ISBN 9780471030027. URL http://books.google.no/books?id=7DAcAAAAIAAJ.

[62] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[63] J. Ashcroft, D. J. Daniels, and S. V. Hart. Forensic examination of digital evidence: A guide for law enforcement. *National Institute of Standards and Technology (NIST) and United States of America*, 2004.

[64] D. Atienza, Á. Herrero, and E. Corchado. Neural analysis of HTTP traffic for web attack detection. In *International Joint Conference*, pages 201–212. Springer, 2015.

[65] J. A. Audestad. *E-Bombs and E-Grenades: The Vulnerability of the Computerized Society*. Gjovik University College, 2011.

[66] T. Aven. *Misconceptions of risk*. John Wiley & Sons, 2011.

[67] J. F. Avila-Herrera and M. M. Subasi. Logical analysis of multi-class data. In *Computing Conference (CLEI), 2015 Latin American*, pages 1–10. IEEE, 2015.

[68] A. Aviles, S. Alsaleh, E. Montseny, P. Sobrevilla, and A. Casals. A deep-neuro-fuzzy approach for estimating the interaction forces in robotic surgery. In *International Conference on Fuzzy Systems (FUZZ-IEEE) 2016*, pages 684–691. Research Publishing Services, 2016.

[69] M. Azimi-Sadjadi, S. Sheedvash, and F. Trujillo. Recursive dynamic node creation in multilayer neural networks. *Neural Networks, IEEE Transactions on*, 4(2):242–256, Mar 1993. ISSN 1045-9227. doi: 10.1109/72.207612.

[70] M. Baig, P. Zavarsky, R. Ruhl, and D. Lindskog. The study of evasion of packed PE from static detection. In *Internet Security (WorldCIS), 2012 World Congress on*, pages 99–104, June 2012.

[71] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, jul 2014. doi: 10.1038/ncomms5308. URL http://dx.doi.org/10.1038/ncomms5308.

[72] A. M. Balogun and S. Y. Zhu. Privacy impacts of data encryption on the efficiency of digital forensics technology. *arXiv preprint arXiv:1312.3183*, 2013.

[73] R. Baltimore. *An Analytic investigation into self organizing maps and their network topologies*. PhD thesis, Rochester Institute of Technology, 2010.

[74] P. Bedi, B. Gupta, and H. Kaur. Access control on grid resources using radial basis function neural network. *Procedia Technology*, 4(0):336 – 341, 2012. ISSN 2212-0173. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[75] N. Beebe. Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics*, pages 17–36. Springer, 2009.

[76] B. Bencsáth. Duqu, Flame, Gauss: Followers of Stuxnet. https://www.rsaconference.com/writable/presentations/file_upload/br-208_bencsath.pdf, 2012. accessed: 10.07.2016.

[77] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

[78] B. Bertoni. Multi-dimensional ellipsoidal fitting. *Department of Physics, South Methodist University, Tech. Rep. SMU-HEP-10-14*, 2010. URL http://www.physics.smu.edu/~scalise/SMUpreprints/SMU-HEP-10-14.pdf.

[79] N. Bezroukov. An overview of malware development history. http://www.softpanorama.org/Malware/Malware_defense_history/Ch01_historic_overview/malware_development_history.shtml, October 2013. accessed: 28.10.2015.

[80] A. Bifet. Data stream mining - regression. accessed: 14.02.2017, May 2012.

[81] A. S. Bist and A. Sharma. Analysis of computer virus using feature fusion. In *2016 Second International Conference on Computational Intelligence Communication Technology (CICT)*, pages 609–614, Feb 2016. doi: 10.1109/CICT.2016.127.

[82] J. J. Blount. *Adaptive rule-based malware detection employing learning classifier systems*. PhD thesis, Missouri University of Science and Technology, 2011.

[83] L. Bottou. On-line learning and stochastic approximations. In D. Saad, editor, *On-Line Learning in Neural Networks*, pages 9–42. Cambridge University Press (CUP). doi: 10.1017/cbo9780511569920.003. URL http://dx.doi.org/10.1017/cbo9780511569920.003.

[84] S. R. Bragen. Malware detection through opcode sequence analysis using machine learning. Master's thesis, Gjøvik University College, 2015.

[85] L. Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[86] B. R. Campomanes-Álvarez, Ó. Cordón, S. Damas, and Ó. Ibáñez. Computer-based craniofacial superimposition in forensic identification using soft computing. *Journal of Ambient Intelligence and Humanized Computing*, 5(5):683–697, 2014. ISSN 1868-5137. doi: 10.1007/s12652-012-0168-1. URL http://dx.doi.org/10.1007/s12652-012-0168-1.

[87] E. Carrera. pefile. https://github.com/erocarrera/pefile, May 2015. accessed: 06.10.2015.

[88] H. Carvey. The Windows registry as a forensic resource. *Digital Investigation*, 2(3):201 – 205, 2005. ISSN 1742-2876.

[89] G. Castellano, A. M. Fanelli, and C. Mencar. Discovering interpretable classification rules from neural processed data. 2002.

[90] C. Cepeda, D. L. C. Tien, and P. Ordóñez. Feature selection and improving classification performance for malware detection. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 560–566, Oct 2016. doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.87.

[91] C.-C. Chang and C.-J. Lin. Training v -support vector regression: Theory and algorithms. *Neural Computation*, 14(8):1959–1977, aug 2002. doi: 10.1162/089976602760128081. URL http://dx.doi.org/10.1162/089976602760128081.

[92] W.-L. Chang, H.-M. Sun, and W. Wu. An Android behavior-based malware detection method using machine learning. In *2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4, Aug 2016. doi: 10.1109/ICSPCC.2016.7753624.

[93] M. Chattopadhyay, P. K. Dan, and S. Mazumdar. Application of visual clustering properties of self organizing map in machine-part cell formation. *Appl. Soft Comput.*, 12(2):600–610, Feb. 2012. ISSN 1568-4946. doi: 10.1016/j.asoc.2011.11.004. URL http://dx.doi.org/10.1016/j.asoc.2011.11.004.

[94] B. Chen. Computer forensics in criminal investigations. *Dujs. dartmouth. edu. Dartmouth Undergraduate Journal of Science*, 13, 2013.

[95] B. Chen. The fundamentals - fuzzy system. Mamdani fuzzy models., September 2013. accessed: 15.08.2015.

[96] C.-H. Chen and K.-C. Li. A three-way classification strategy for reducing class-abundance: The zip code recognition example. *Lecture Notes-Monograph Series*, pages 63–86, 2004.

[97] C.-K. Chen. Malware classification and detection. http://www.slideshare.net/Bletchley131/malware-classificationanddetection, May 2015. accessed: 10.07.2016.

[98] J. Y.-C. Cheng, T.-S. Tsai, and C.-S. Yang. An information retrieval approach for malware classification based on Windows API calls. In *2013 International Conference on Machine Learning and Cybernetics*, volume 04, pages 1678–1683, July 2013. doi: 10.1109/ICMLC.2013.6890868.

[99] D. M. Chess and S. R. White. An undetectable computer virus. In *Proceedings of Virus Bulletin Conference*, volume 5, 2000.

[100] E. Chester. Are the terms "hard work" and "work ethic" synonymous?sigkdd. http://www.revivingworkethic.com/terms-hard-work-work-ethic-synonymous/, September 2011. accessed: 15.12.2014.

[101] A. S. Chitrakar and K. Franke. Author identification from text-based communications: Identifying generalized features and computational methods. *Norsk informasjonssikkerhetskonferanse (NISK)*, 2013, 2014.

[102] M. G. Christian Funk. Kaspersky security bulletin 2013. overall statistics for 2013. https://securelist.com/analysis/kaspersky-security-bulletin/58265/kaspersky-security-bulletin-2013-overall-statistics-for-2013/, December 2013. accessed: 15.10.2015.

[103] M. Clark. A comparison of correlation measures. Technical report, University of Notre Dame, 2013.

[104] F. Cohen. Computer viruses: theory and experiments. *Computers & security*, 6(1):22–35, 1987.

[105] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *Decis. Support Syst.*, 47(4):547–553, Nov. 2009. ISSN 0167-9236.

[106] M. Cottrell, M. Olteanu, F. Rossi, J. Rynkiewicz, and N. Villa-Vialaneix. Neural networks for complex data. *KI - Künstliche Intelligenz*, 26(4):373–380, 2012. ISSN 0933-1875.

[107] S. D, D. F. Kibler, M. J. Pazzani, and P. Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Explorations*, 2:81, 2000.

[108] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-scale malware classification using random projections and neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3422–3426. IEEE, 2013.

[109] S. Damas, O. Cordón, O. Ibáñez, J. Santamaría, I. Alemán, M. Botella, and F. Navarro. Forensic identification by computer-aided craniofacial superimposition: A survey. *ACM Comput. Surv.*, 43(4):27:1–27:27, Oct. 2011. ISSN 0360-0300. doi: 10.1145/1978802.1978806. URL http://doi.acm.org/10.1145/1978802.1978806.

[110] D. Danchev. Why relying on antivirus signatures is simply not enough anymore. http://blog.webroot.com/2012/02/23/why-relying-on-antivirus-signatures-is-simply-not-enough-anymore/. Accessed: 09.04.2013.

[111] S. K. Das, A. Kumar, B. Das, and A. Burnwal. On soft computing techniques in various areas. *Computer Science & Information Technology*, page 59, 2013.

[112] G. De Tré, J. Nielandt, A. Bronselaer, D. Vandermeulen, J. Hermans, and P. Claeys. LSP based comparison of 3D ear models. In *Norbert Wiener in the 21st Century (21CW), 2014 IEEE Conference on*, pages 1–7. IEEE, 2014.

[113] O. De Vel, A. Anderson, M. Corney, and G. Mohay. Mining e-mail content for author identification forensics. *ACM Sigmod Record*, 30(4):55–64, 2001.

[114] P. M. Deepika Veerwal. Ensemble of soft computing techniques for malware detection. *International Journal of Emerging Technologies in Computational and Applied Sciences*, 6:159–167, September-November 2013.

[115] R. DeLisle. Kohonen's self organizing feature maps. http://www.ai-junkie.com/ann/som/som1.html. accessed: 14.02.2017.

[116] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, machine learning can be more secure! a case study on Android malware detection. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2017. ISSN 1545-5971. doi: 10.1109/TDSC.2017.2700270.

[117] Y. DENG, Z. Ren, Y. Kong, F. Bao, and Q. Dai. A hierarchical fused fuzzy deep neural network for data classification. *IEEE Transactions on Fuzzy Systems*, PP(99):1–1, 2016. ISSN 1063-6706. doi: 10.1109/TFUZZ.2016.2574915.

[118] D. Devi and S. Nandi. Detection of packed malware. In *Proceedings of the First International Conference on Security of Internet of Things*, SecurIT '12, pages 22–26, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1822-8. doi: 10.1145/2490428.2490431. URL http://doi.acm.org/10.1145/2490428.2490431.

[119] A. K. Dey and D. Kundu. Discriminating between the log-normal and log-logistic distributions. *Communications in Statistics-Theory and Methods*, 39(2):280–292, 2009.

[120] J. Dickerson and B. Kosko. Fuzzy function approximation with ellipsoidal rules. *IEEE Trans. Syst., Man, Cybern. B*, 26(4):542–560, 1996. doi: 10.1109/3477.517030. URL http://dx.doi.org/10.1109/3477.517030.

[121] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, pages 263–286, 1995.

[122] S. Ding, H. Li, C. Su, J. Yu, and F. Jin. Evolutionary artificial neural networks: a review. *Artificial Intelligence Review*, 39(3):251–260, 2013. ISSN 1573-7462.

[123] D. Distler and C. Hornat. Malware analysis: An introduction. *Sans Reading Room*, 2007.

[124] B. Dolan-Gavitt. Forensic analysis of the Windows registry in memory. *Digital Investigation*, 5, Supplement:S26 – S32, 2008. ISSN 1742-2876. The Proceedings of the Eighth Annual {DFRWS} Conference.

[125] B. Dolan-Gavitt, A. Srivastava, P. Traynor, and J. Giffin. Robust signatures for kernel data structures. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 566–577. ACM, 2009.

[126] J. Dolcourt. Best phones of 2016. https://www.cnet.com/topics/phones/best-phones/, December 2016. accessed: 20.12.2016.

[127] R. B. Doorenbos. *Production Matching for Large Learning Systems*. PhD thesis, Pittsburgh, PA, USA, 1995. UMI Order No. GAX95-22942.

[128] M. Drobics, W. Winiwater, and U. Bodenhofer. Interpretation of self-organizing maps with fuzzy rules. In *Tools with Artificial Intelligence, 2000. ICTAI 2000. Proceedings. 12th IEEE International Conference on*, pages 304–311, 2000. doi: 10.1109/TAI.2000.889887.

[129] P. Duan and B. Solecki. Solution for the Amazon employee access challenge. https://github.com/pyduan/amazonaccess. accessed: 10.12.2015.

[130] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers. Malware type recognition and cyber situational awareness. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 938–943, Aug 2010. doi: 10.1109/SocialCom.2010.139.

[131] S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, jun 2003. doi: 10.1093/bioinformatics/btg038. URL http://dx.doi.org/10.1093/bioinformatics/btg038.

[132] R. G. Dukhi. Soft computing tools in credit card fraud & detection, 2011.

[133] T. Ebringer, L. Sun, and S. Boztas. A fast randomness test that preserves local detail. *Virus Bulletin*, 2008, 2008.

[134] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.*, 44 (2):6:1–6:42, Mar. 2008. ISSN 0360-0300.

[135] N. Eiamkanitchat, N. Theera-Umpon, and S. Auephanwiriyakul. A novel neuro-fuzzy method for linguistic feature selection and rule-based classification. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 2, pages 247–252, Feb 2010.

[136] G. Erdélyi. Malware taxonomy. accessed: 16.02.2017, 2010.

[137] P. Estévez, J. Príncipe, and P. Zegers. *Advances in Self-Organizing Maps: 9th International Workshop, WSOM 2012 Santiago, Chile, December 12-14, 2012 Proceedings*. Advances in Intelligent Systems and Computing. Springer, 2012. ISBN 9783642352300. URL https://books.google.no/books?id=vHgnfKFpIFUC.

[138] Europol. Mobile malware. https://www.europol.europa.eu/activities-services/public-awareness-and-prevention-guides/mobile-malware. accessed:03.05.2017.

[139] G. R. Exner. *Inside Calculus*. Undergraduate Texts in Mathematics. Springer-Verlag, Springer-Verlag, 2000.

[140] EY. Forensic data analytics. http://www.ey.com/Publication/vwLUAssets/EY_-_Forensic_Data_Analytics_(FDA)/$FILE/EY-forensic-data-dnalytics.pdf, 2013. accessed: 20.12.2016.

[141] EY. Forensic data analytics. http://www.ey.com/Publication/vwLUAssets/EY_-_Forensic_Data_Analytics_(FDA)/$FILE/EY-forensic-data-dnalytics.pdf, 2013. accesed: 16.08.2016.

[142] D. Farber and R. Lachman. Similarity-based access control of data in a data processing system, May 17 2011. US Patent 7,945,544.

[143] P. Faruki, V. Laxmi, M. S. Gaur, and P. Vinod. Mining control flow graph as API call-grams to detect portable executable malware. In *Proceedings of the Fifth International Conference on Security of Information and Networks*,

SIN '12, pages 130–137, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1668-2. doi: 10.1145/2388576.2388594. URL http://doi.acm.org/10.1145/2388576.2388594.

[144] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[145] J. fei Qiao and H. gui Han. *An Adaptive Fuzzy Neural Network Based on Self-Organizing Map (SOM)*. INFTECH, April 2010. ISBN 978-953-307-074-2.

[146] E. R. Feldman. Criteria for admissibility of expert opinion testimony under daubert and its progeny. Technical report, Cozen O'Connor, 2001.

[147] ForensicsWiki. Digital evidence. http://forensicswiki.org/wiki/Digital_evidence. accessed: 06.02.2017.

[148] T. Fox-Brewster. Yahoo: Hackers stole data on another billion accounts – updated. http://www.forbes.com/sites/thomasbrewster/2016/12/14/yahoo-admits-another-billion-user-accounts-were-leaked-in-2013/, December 2016. accessed: 20.12.2016.

[149] E. Frank, M. Hall, P. Reutemann, and L. Trigg. Weka 3: Data mining software in Java. http://www.cs.waikato.ac.nz/ml/weka/. accessed: 10.4.2015.

[150] K. Franke. *The Influence of Physical and Biomechanical Processes on the Ink Trace: Methodological Foundations for the Forensic Analysis of Signatures*. Rijksuniv., 2005. ISBN 9783000173639. URL https://books.google.no/books?id=wSxmAAAACAAJ.

[151] K. Franke and S. N. Srihari. *Computational Forensics: An Overview*, pages 1–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-85303-9. doi: 10.1007/978-3-540-85303-9_1. URL http://dx.doi.org/10.1007/978-3-540-85303-9_1.

[152] K. Franke, Y.-N. Zhang, and M. Köppen. Static signature verification employing a Kosko-Neuro-Fuzzy approach. In N. Pal and M. Sugeno, editors, *Advances in Soft Computing — AFSS 2002*, volume 2275 of *Lecture Notes in Computer Science*, pages 185–190. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43150-3. doi: 10.1007/3-540-45631-7_26. URL http://dx.doi.org/10.1007/3-540-45631-7_26.

[153] R. Freund, D. Mohr, and W. Wilson. *Statistical Methods*. Elsevier Science, 2010. ISBN 9780080961033. URL https://books.google.no/books?id=12LPWl6QxrsC.

[154] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[155] R. Fuller. *Introduction to Neuro-Fuzzy Systems*. Advances in Intelligent and Soft Computing. Physica-Verlag HD, 2000. ISBN 9783790812565. URL http://books.google.no/books?id=1ihxRjLGGIcC.

[156] M. Gacto, R. Alcalá, and F. Herrera. Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures. *Information Sciences*, 181(20):4340 – 4360, 2011. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2011.02.021. URL http://www.sciencedirect.com/science/article/pii/S0020025511001034. Special Issue on Interpretable Fuzzy Systems.

[157] B. Gallagher and T. Eliassi-Rad. Classification of http attacks: a study on the ECML/PKDD 2007 discovery challenge. In *Center for Advanced Signal and Image Sciences (CASIS) Workshop*, 2008.

[158] E. Gandotra, D. Bansal, and S. Sofat. Malware analysis and classification: A survey. *Journal of Information Security*, 2014, 2014.

[159] S. L. Garfinkel. Digital forensics research: The next 10 years. *digital investigation*, 7:S64–S73, 2010.

[160] D. Gavrilut, M. Cimpoesu, D. Anton, and L. Ciortuz. Malware detection using machine learning. In *Computer Science and Information Technology, 2009. IMCSIT '09. International Multiconference on*, pages 735–741, Oct 2009.

[161] R. Gentleman, R. Ihaka, D. Bates, et al. The R project for statistical computing. http://www.r-project.org, 1997. accessed: 15.02.16.

[162] M. Geraily and M. V. Jahan. Fuzzy detection of malicious attacks on web applications based on Hidden Markov Model ensemble. In *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pages 102–108. IEEE, 2012.

[163] T. Germano. Self Organizing Maps. http://davis.wpi.edu/~matt/courses/soms/. accessed: 14.02.2017.

[164] Z. Ghahramani. Probabilistic modelling, machine learning, and the information revolution. In *presentation at MIT CSAIL*, 2012.

[165] Google. App manifest. https://developer.android.com/guide/topics/manifest/uses-sdk-element.html.

[166] S. Gordon and R. Ford. On the definition and classification of cybercrime. *Journal in Computer Virology*, 2(1):13–20, 2006. ISSN 1772-9904. doi: 10.1007/s11416-006-0015-z. URL http://dx.doi.org/10.1007/s11416-006-0015-z.

[167] L. S. Grini, A. Shalaginov, and K. Franke. Study of soft computing methods for large-scale multinomial malware types and families detection. In *The 6th World Conference on Soft Computing*, 2016.

[168] C. Grobler and C. Louwrens. Digital forensic readiness as a component of information security best practice. *New approaches for security, privacy and trust in complex environments*, pages 13–24, 2007.

[169] A. Guarino. Digital Forensics as a Big Data Challenge. In *ISSE 2013 Securing Electronic Business Processes*, pages 197–203. Springer, 2013.

[170] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser. The cuckoo Sandbox, 2012.

[171] S. Guillaume. Designing fuzzy inference systems from data: An interpretability-oriented review. *Trans. Fuz Sys.*, 9(3):426–443, June 2001. ISSN 1063-6706. doi: 10.1109/91.928739. URL http://dx.doi.org/10.1109/91.928739.

[172] N. R. Guo, C.-L. Kuo, and T.-J. Tsai. Design of an EP-based neuro-fuzzy classification model. In *Networking, Sensing and Control, 2009. ICNSC '09. International Conference on*, pages 918–923, March 2009.

[173] K. Hahn. Robust static analysis of portable executable malware. Mater thesis, HTWK Leipzig, 2014.

[174] W. Halboob, R. Mahmod, N. I. Udzir, and M. T. Abdullah. Privacy levels for computer forensics: Toward a more efficient privacy-preserving investigation. *Procedia Computer Science*, 56:370 – 375, 2015. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2015.07.222. URL http://www.sciencedirect.com/science/article/pii/S1877050915017032.

[175] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[176] G. Hallevy. *Liability for Crimes Involving Artificial Intelligence Systems*. Springer International Publishing, Cham. ISBN 9783319101248. URL https://books.google.no/books?id=xf05BQAAQBAJ.

[177] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[178] R. M. Harris. Using artificial neural networks for forensic file type identification. *Master's Thesis, Purdue University*, 2007.

[179] P. Harvey. ExifTool. http://www.sno.phy.queensu.ca/~phil/exiftool/, November 2015. accessed: 04.11.2015.

[180] S. Hasan and S. M. Shamsuddin. Multistrategy self-organizing map learning for classification problems. *Computational Intelligence and Neuroscience*, 2011:11, 2011.

[181] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4): 18–28, 1998.

[182] J. L. Hernandez, M. Moreno, A. Jara, and A. F. Skarmeta. A soft computing based location-aware access control for smart buildings. *Soft Computing*, 18 (9):1659–1674, 2014. ISSN 1432-7643. doi: 10.1007/s00500-014-1278-9. URL http://dx.doi.org/10.1007/s00500-014-1278-9.

[183] L. Herrera, H. Pomares, I. Rojas, O. Valenzuela, and A. Prieto. Tase, a taylor series-based fuzzy system model that combines interpretability and accuracy. *Fuzzy Sets and Systems*, 153(3):403 – 427, 2005. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/j.fss.2005.01.012. URL http://www.sciencedirect.com/science/article/pii/S0165011405000333.

[184] T. M. Heskes and B. Kappen. On-line learning processes in artificial neural networks, 1993.

[185] Hex-Rays. IDA Pro. https://www.hex-rays.com/products/ida/. accessed: 12.10.2015.

[186] J. Hollmen. Self-Organizing Map (SOM). http://users.ics.aalto.fi/jhollmen/dippa/node9.html, March 1996. accessed: 14.02.2017.

[187] Y. Hong, C. Huang, B. Nandy, and N. Seddigh. Iterative-tuning support vector machine for network traffic classification. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 458–466, May 2015.

[188] V. C. Hu, D. Ferraiolo, Rick, Schnitzer, Adam, Sandlin, Kenneth, Miller, Robert, Scarfone, and K. Scarfone. Guide to attribute based access control (ABAC) definition and considerations. Guide, National Institute of Standards and Technology, 2014.

[189] Z. Hu, Y. V. Bodyanskiy, and O. K. Tyshchenko. A deep cascade neuro-fuzzy system for high-dimensional online fuzzy clustering. In *2016 IEEE First International Conference on Data Stream Mining Processing (DSMP)*, pages 318–322, Aug 2016. doi: 10.1109/DSMP.2016.7583567.

[190] S.-Y. Huang and Y. Huang. Network forensic analysis using growing hierarchical SOM. In *2013 IEEE 13th International Conference on Data Mining Workshops*. Institute of Electrical & Electronics Engineers (IEEE), dec 2013. doi: 10.1109/icdmw.2013.66. URL http://dx.doi.org/10.1109/icdmw.2013.66.

[191] S.-Y. Huang and Y.-N. Huang. Network traffic anomaly detection based on growing hierarchical SOM. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Institute of Electrical & Electronics Engineers (IEEE), jun 2013. doi: 10.1109/dsn.2013.6575338. URL http://dx.doi.org/10.1109/dsn.2013.6575338.

[192] J. S. Hurwitz. *Error-correcting codes and applications to large scale classification systems*. PhD thesis, Massachusetts Institute of Technology, 2009.

[193] R. Husain and S. Muhammad. A survey on soft computing techniques in network security. *Scholarly Journal of Mathematics and Computer Science*, 2(3):28–32, June 2013.

[194] I. Iancu. A mamdani type fuzzy logic controller. http://cdn.intechopen.com/pdfs-wm/34221.pdf. accessed: 23.09.2014.

[195] O. Ibanez, L. Ballerini, O. Cordon, S. Damas, and J. Santamaria. An experimental study on the applicability of evolutionary algorithms to craniofacial superimposition in forensic identification. *Information Sciences*, 179 (23):3998 – 4028, 2009. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2008.12.029. URL http://www.sciencedirect.com/science/article/pii/S0020025509000085.

[196] N. Idika and A. P. Mathur. A survey of malware detection techniques. *Purdue University*, 48, 2007.

[197] Interpol. Cybercrime. https://www.interpol.int/Crime-areas/Cybercrime/Cybercrime. accessed: 15.01.2017.

[198] H. Ishibuchi and T. Nakashima. Effect of rule weights in fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, pages 260–270, 2001.

[199] H. Ishibuchi and Y. Nojima. Discussions on interpretability of fuzzy systems using simple examples, 2009.

[200] H. Ishibuchi, T. Yamamoto, S. Member, P. H. Ishibuchi, H. Ishibuchi, T. Yamamoto, and S. Member. Rule weight specification in fuzzy rule-based classification systems. *IEEE Trans. Fuzzy Syst*, 13:428–435, 2005.

[201] G. Jacob, H. Debar, and E. Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4 (3):251–266, 2008. ISSN 1772-9904.

[202] A. K. Jain, R. P. Duin, and J. Mao. Statistical pattern recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1): 4–37, 2000.

[203] S. Jain and Y. K. Meena. Byte level n–gram analysis for malware detection. In *Computer Networks and Intelligent Computing*, pages 51–59. Springer, 2011.

[204] J. I. James and P. Gladyshev. Challenges with automation in digital forensic investigations. http://arxiv.org/pdf/1303.4498.pdf. accessed: 11.12.2014.

[205] Q. Jamil and M. A. Shah. Analysis of machine learning solutions to detect malware in Android. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 226–232, Aug 2016. doi: 10.1109/INTECH.2016.7845073.

[206] W. Jansen, R. Ayers, and S. Brothers. Guidelines on mobile device forensics. *NIST Special Publication*, pages 800–101, 2014.

[207] C. J.Geyer. Stat 5102 notes: More on confidence intervals. http://www.stat.umn.edu/geyer/old03/5102/notes/ci.pdf, February 2003. accessed: 07.04.2015.

[208] Z. Jie, H. Long, and R. Sijing. RBF neural network adaptive sliding mode control based on genetic algorithm optimization. In *2016 Chinese Control and Decision Conference (CCDC)*, pages 6772–6775, May 2016.

[209] Y. Jin. Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement. *Fuzzy Systems, IEEE Transactions on*, 8 (2):212–221, Apr 2000. ISSN 1063-6706. doi: 10.1109/91.842154.

[210] S. Kadirvelu and K. Arputharaj. Handling web and database requests using fuzzy rules for anomaly intrusion detection. *Journal of Computer Science*, 7(2):255, 2011.

[211] Kaggle. Amazon.com - employee access challenge. https://www.kaggle.com/c/amazon-employee-access-challenge, May 2013. accessed: 20.05.2016.

[212] E. Kamar. Directions in hybrid intelligence: complementing ai systems with human intelligence. *Early Career Track*, 2016.

[213] Y. Kanada. Optimizing neural-network learning rate by using a genetic algorithm with per-epoch mutations. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1472–1479, July 2016.

[214] N. Kandil, K. Khorasani, R. Patel, and V. Sood. Optimum learning rate for backpropagation neural networks. In *Electrical and Computer Engineering, 1993. Canadian Conference on*, pages 465–468 vol.1, Sep 1993.

[215] A. Karpathy, F. Li, and J. Johnson. CS231n Convolutional neural network for visual recognition. *Online Course*, 2016.

[216] F. O. Karray and C. W. De Silva. *Soft computing and intelligent systems design: theory, tools, and applications*. Pearson Education, 2004.

[217] kaspersky. Types of malware. http://www.kaspersky.com/internet-security-center/threats/types-of-malware. accessed: 06.07.2016.

[218] R. Kath. The Portable Executable file format from top to bottom. *MSDN Library, Microsoft Corporation*, 1993.

[219] M. H. B. Katherine L Milkman, Dolly Chugh. How can decision making be improved? *Perspectives on Psychological Science*, 4(4):379–383, July 2009.

[220] A. Kaur and A. Kaur. Comparison of fuzzy logic and neuro-fuzzy algorithms for air conditioning system.

[221] K. Kendall and C. McMillan. Practical malware analysis. In *Black Hat Conference, USA*, 2007.

[222] M. Kheirkhahan, M. Yashtini, A. Youse, and H. Wehry. CAP6610 Machine Learning Project. Technical report, Georgia Institute of Technology, School of Mathematics, 2014.

[223] Z. Khorsand and A. Hamzeh. A novel compression-based approach for malware detection using PE header. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 127–133, May 2013. doi: 10.1109/IKT.2013.6620051.

[224] H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park. Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. *Neurocomputing*, 11(1):101 – 106, may 1996. ISSN 0925-2312.

[225] H. M. Kim and J. Mendel. Fuzzy basis functions: comparisons with other basis functions. *IEEE Trans. Fuzzy Syst.*, 3(2):158–168, may 1995. doi: 10.1109/91.388171. URL http://dx.doi.org/10.1109/91.388171.

[226] J.-S. Kim, D.-G. Kim, and B.-N. Noh. A fuzzy logic based expert system as a network forensics. In *IEEE International Conference on Fuzzy Systems (IEEE Cat. No.04CH37542)*. Institute of Electrical & Electronics Engineers (IEEE), 2004. doi: 10.1109/fuzzy.2004.1375521. URL http://dx.doi.org/10.1109/fuzzy.2004.1375521.

[227] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[228] B. Klimt and Y. Yang. Introducing the enron corpus. In *CEAS*, 2004.

[229] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982. ISSN 1432-0770. doi: 10.1007/BF00337288. URL http://dx.doi.org/10.1007/BF00337288.

[230] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 470–478, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014105. URL http://doi.acm.org/10.1145/1014052.1014105.

[231] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.*, 7:2721–2744, Dec. 2006. ISSN 1532-4435.

[232] I. Kononenko and M. Kukar. *Machine learning and data mining: introduction to principles and algorithms*. Horwood Publishing, 2007.

[233] B. Kosko. *Fuzzy Engineering*. Number v. 1 in Fuzzy Engineering. Prentice Hall, 1997. ISBN 9780131249912. URL http://books.google.no/books?id=8QwoAQAAMAAJ.

[234] A. Kramer. Review of Windows 7 as a malware analysis environment. *Sans Reading Room*, 2014.

[235] R. Kruse. Fuzzy neural network. *Scholarpedia*, 3(11):6043, 2008.

[236] K.-C. Kwak. An incremental adaptive neuro-fuzzy networks. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 1407–1410, Oct 2008. doi: 10.1109/ICCAS.2008.4694363.

[237] O. Köksoy and T. Yalcinoz. Robust design using pareto type optimization: A genetic algorithm with arithmetic crossover. *Computers & Industrial Engineering*, 55(1):208 – 218, aug 2008. ISSN 0360-8352.

[238] A. D. Landress. A hybrid approach to reducing the false positive rate in unsupervised machine learning intrusion detection. In *SoutheastCon 2016*, pages 1–6, March 2016. doi: 10.1109/SECON.2016.7506773.

[239] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *IEEE Communications magazine*, 48(9):140–150, 2010.

[240] Lastline. The threat of evasive malware. White paper, Lastline Labs, https://www.lastline.com/papers/evasive_threats.pdf, February 2013. accessed: 29.10.2015.

[241] R. Lau, Y. Xia, and Y. Ye. A probabilistic generative model for mining cybercriminal networks from online social media. *Computational Intelligence Magazine, IEEE*, 9(1):31–43, Feb 2014. ISSN 1556-603X. doi: 10.1109/MCI.2013.2291689.

[242] F. Y. W. Law, P. P. F. Chan, S. M. Yiu, K. P. Chow, M. Y. K. Kwan, H. K. S. Tse, and P. K. Y. Lai. Protecting digital data privacy in computer forensic examination. In *2011 Sixth IEEE International Workshop on Systematic*

Approaches to Digital Forensic Engineering, pages 1–6, May 2011. doi: 10.1109/SADFE.2011.15.

[243] N. A. Le-Khac and A. Linke. Control flow change in assembly as a classifier in malware analysis. In *2016 4th International Symposium on Digital Forensic and Security (ISDFS)*, pages 38–43, April 2016. doi: 10.1109/ISDFS.2016.7473514.

[244] B. LeBlanc. 64-bit momentum surges with Windows 7. https://blogs.windows.com/windowsexperience/2010/07/08/64-bit-momentum-surges-with-windows-7/, July 2010. accessed: 15.10.2015.

[245] H. C. Lee and E. M. Pagliaro. Forensic evidence and crime scene investigation. *Journal of Forensic Investigation*, 01(02), 2013. doi: 10.13188/2330-0396.1000004. URL http://dx.doi.org/10.13188/2330-0396.1000004.

[246] M. Leeds and T. Atkison. Preliminary results of applying machine learning algorithms to Android malware detection. In *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 1070–1073, Dec 2016. doi: 10.1109/CSCI.2016.0204.

[247] W. Leonhard. ATMs will still run Windows XP – but a bigger shift in security looms. http://www.infoworld.com/article/2610392/microsoft-windows/atms-will-still-run-windows-xp----but-a-bigger-shift-in-security-looms.html, March 2014. accessed: 09.11.2015.

[248] S. Lian, S. Gritzalis, N. Nedjah, and I.-C. Lin. Special issue on soft computing for information system security. *Applied Soft Computing*, 11(7): 4257 – 4259, 2011. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2011.05.040. URL http://www.sciencedirect.com/science/article/pii/S1568494611002006. Soft Computing for Information System Security.

[249] N. Liao, S. Tian, and T. Wang. Network forensics based on fuzzy logic and expert system. *Computer Communications*, 32(17):1881 – 1892, 2009. ISSN 0140-3664. doi: http://dx.doi.org/10.1016/j.comcom.2009.07.013. URL http://www.sciencedirect.com/science/article/pii/S0140366409002060.

[250] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[251] M. Ligh, S. Adair, B. Hartstein, and M. Richard. *Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code*. Wiley Publishing, 2010.

[252] C.-t. Lin. *An Efficient Feature Selection and Extraction Analysis for Malware Behavior Classification*. PhD thesis, 2015.

[253] Y.-H. Lin and M.-S. Tsai. Non-intrusive load monitoring by novel neuro-fuzzy classification considering uncertainties. *Smart Grid, IEEE Transactions on*, 5(5):2376–2384, Sept 2014. ISSN 1949-3053.

[254] Z. Lin, J. Rhee, X. Zhang, D. Xu, and X. Jiang. SigGraph: Brute force scanning of kernel data structure instances using graph-based signatures. In *NDSS*, 2011.

[255] M. Liu, D. Zhang, S. Chen, and H. Xue. Joint binary classifier learning for ECOC-based multi-class classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1–1, 2015. ISSN 0162-8828.

[256] D. G. Luenberger and Y. Ye. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[257] E. Lughofer. *Evolving Fuzzy Systems - Methodologies, Advanced Concepts and Applications*. Studies in Fuzziness and Soft Computing. Springer, 2011. ISBN 9783642180866. URL http://books.google.no/books?id=CP0qmaGuZf0C.

[258] J. R. Lyle, D. R. White, and R. P. Ayers. Digital forensics at the national institute of standards and technology. *National Institute of Standards and Technology, Interagency Report (NISTIR)*, 7490, 2008.

[259] G. Madzarov and D. Gjorgjevikj. Multi-class classification using support vector machines in decision tree architecture. In *IEEE EUROCON 2009*, pages 288–295, May 2009.

[260] F. Maiorana, N. Mastorakis, M. Poulos, V. Mladenov, Z. Bojkovic, D. Simian, S. Kartalopoulos, A. Varonides, and C. Udriste. Performance improvements of a kohonen self organizing classification algorithm on sparse data sets. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*, number 10. WSEAS, 2008.

[261] A. Makiou, Y. Begriche, and A. Serhrouchni. Hybrid approach to detect SQLi attacks and evasion techniques. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*, pages 452–456, Oct 2014.

[262] D. P. Mandic and J. A. Chambers. Towards the optimal learning rate for backpropagation. *Neural Process. Lett.*, 11(1):1–5, Feb. 2000. ISSN 1370-4621. doi: 10.1023/A:1009686825582.

[263] T. C. K. Mark L. Berenson, David M. Levine. *Basic Business Statistics, 11/E*. Pearson, 2009.

[264] Z. Markel and M. Bilzor. Building a machine learning classifier for malware detection. In *Anti-malware Testing Research (WATeR), 2014 Second Workshop on*, pages 1–4. IEEE, 2014.

[265] J. Marpaung, M. Sain, and H.-J. Lee. Survey on malware evasion techniques: State of the art and challenges. In *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, pages 744 –749, feb. 2012.

[266] B. Marr. Why only one of the 5 Vs of big data really matters. http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters, March 2015. accessed: 23.12.2016.

[267] E. Martínez-Gómez, M. T. Richards, and D. S. P. Richards. Distance correlation methods for discovering associations in large astrophysical databases. *The Astrophysical Journal*, 781:39, Jan. 2014. doi: 10.1088/0004-637X/781/1/39.

[268] M. Masud, L. Khan, and B. Thuraisingham. A hybrid model to detect malicious executables. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1443–1448, June 2007. doi: 10.1109/ICC.2007.242.

[269] K. Maxwell. Maltrieve - a tool to retrieve malware directly from the source for security researchers. , May 2015. accessed: 06.10.2015.

[270] McAfee. Mobile threat report. what's on the horizon for 2016. http://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf, August 2016. accesed: 20.12.2016.

[271] McAfee. Part of Intel Security. Threats report. Technical report, McAfee, August 2015. accessed: 19.09.2015.

[272] D. McAllester. Neural networks: Backpropagation general gradient descent. http://ttic.uchicago.edu/~dmcallester/ttic101-07/lectures/neural/neural.pdf. University of Chicago.

[273] J. L. McClelland and D. E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-63113-X.

[274] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, nov 2000. doi: 10.1145/382912.382923. URL https://doi.org/10.1145%2F382912.382923.

[275] R. McKemmish. When is digital evidence forensically sound? In *IFIP — The International Federation for Information Processing*, pages 3–15. Springer Science+Business Media, 2008. doi: 10.1007/978-0-387-84927-0_1. URL http://dx.doi.org/10.1007/978-0-387-84927-0_1.

[276] E. Menahem, A. Shabtai, L. Rokach, and Y. Elovici. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, 53(4):1483 – 1494, 2009. ISSN 0167-9473.

[277] Miccrosoft. Windows registry information for advanced users. https://support.microsoft.com/en-us/kb/256986. accessed: 12.07.2016.

[278] Microsoft. Malware families cleaned by the malicious software removal tool. https://www.microsoft.com/security/pc-security/malware-families.aspx, .

[279] Microsoft. A history of Windows. http://windows.microsoft.com/en-us/windows/history, . accessed: 08.01.2016.

[280] Microsoft. The Microsoft Windows malicious software removal tool helps remove specific, prevalent malicious software from computers that are running supported versions of windows. https://support.microsoft.com/en-us/kb/890830, July 2016. accessed: 15.07.2016.

[281] Microsoft Malware Protection Center. Naming malware.

[282] N. Milosevic. History of malware. *arXiv preprint arXiv:1302.5392*, 2013.

[283] A. A. Minai and R. D. Williams. On the derivatives of the sigmoid. *Neural Networks*, 6(6):845 – 853, 1993. ISSN 0893-6080.

[284] T. M. Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45: 37, 1997.

[285] S. Mitra, S. K. Pal, and P. Mitra. Data mining in soft computing framework: a survey. *IEEE transactions on neural networks*, 13(1):3–14, 2002.

[286] D. Mo. A survey on deep learning: one small step toward AI. *Dept. Computer Science, Univ. of New Mexico, USA*, 2012.

[287] A. Modi, Z. Sun, A. Panwar, T. Khairnar, Z. Zhao, A. Doupé, G. J. Ahn, and P. Black. Towards automated threat intelligence fusion. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 408–416, Nov 2016. doi: 10.1109/CIC.2016.060.

[288] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421 –430, dec. 2007. doi: 10.1109/ACSAC.2007.21.

[289] A. K. Muda, Y.-H. Choo, A. Abraham, and S. N. Srihari, editors. *Computational Intelligence in Digital Forensics: Forensic Investigation and Applications*. Springer International Publishing, 2014. doi: 10.1007/978-3-319-05885-6. URL http://dx.doi.org/10.1007/978-3-319-05885-6.

[290] S. Mukkamala and A. H. Sung. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence*, 1(4):1–17, 2003.

[291] A. Mushtaq. World's top malware. https://www.fireeye.com/blog/threat-research/2010/07/worlds_top_modern_malware.html, July 2010. accessed: 15.07.2016.

[292] S. Naaz, A. Alam, and R. Biswas. Effect of different defuzzification methods in a fuzzy based load balancing application. *IJCSI-International Journal of Computer Science Issues*, 8(5), 2011.

[293] O. Nelles, M. Fischer, and B. Muller. Fuzzy rule extraction by a genetic algorithm and constrained nonlinear optimization of membership functions. In *Fuzzy Systems, 1996., Proceedings of the Fifth IEEE International Conference on*, volume 1, pages 213–219 vol.1, Sep 1996. doi: 10.1109/FUZZY.1996.551744.

[294] H. Nguyen, K. Franke, and S. Petrovic. Improving effectiveness of intrusion detection by correlation feature selection. In *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*, pages 17–24. IEEE, 2010.

[295] L. A. T. Nguyen and H. K. Nguyen. Phishing identification: An efficient neuro-fuzzy model without using rule sets. In *Control Conference (ASCC), 2015 10th Asian*, pages 1–6, May 2015.

[296] Norton. What is cybercrime? https://us.norton.com/cybercrime-definition. accessed: 17.01.2017.

[297] L. Null and J. Lobur. *The Essentials of Computer Organization and Architecture*. Jones and Bartlett Publishers, Inc., USA, 4th edition, 2014. ISBN 1284045617, 9781284045611.

[298] I. L. Nunes and M. Simões-Marques. *Applications of Fuzzy Logic in Risk Assessment - The RA_X Case*. INTECH Open Access Publisher, 2012.

[299] K. K. Oad, X. DeZhi, and P. K. Butt. A fuzzy rule based approach to predict risk level of heart disease. *Global Journal of Computer Science and Technology*, 14(3), 2014.

[300] U. D. of Homeland Security. National software reference library project. http://www.nsrl.nist.gov/. accessed: 29.11.2015.

[301] C. Olah. Neural networks, manifolds, and topology. http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/, March 2014. accessed: 03.11.2016.

[302] OSDev. PE - Portable executable. http://wiki.osdev.org/PE. accessed: 10.10.2015.

[303] G. Ou and Y. L. Murphey. Multi-class pattern classification using neural networks. *Pattern Recogn.*, 40(1):4–18, Jan. 2007. ISSN 0031-3203.

[304] S. J. Ovaska, H. F. VanLandingham, and A. Kamiya. Fusion of soft computing and hard computing in industrial applications: An overview. *Trans. Sys. Man Cyber Part C*, 32(2):72–79, May 2002. ISSN 1094-6977. doi: 10.1109/TSMCC.2002.801354. URL http://dx.doi.org/10.1109/TSMCC.2002.801354.

[305] T. OWASP. Top 10–2013. *The Ten Most Critical Web Application Security Risks*, 2013.

[306] K. Pachopoulos, D. Valsamou, D. Mavroeidis, and M. Vazirgiannis. Feature extraction from web traffic data for the application of data mining algorithms in attack identification. In *Proceedings of the ECML/PKDD*, pages 65–70, 2007.

[307] S. Pal, A. Petrosino, and L. Maddalena. *Handbook on Soft Computing for Video Surveillance*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2012. ISBN 9781439856840. URL http://books.google.no/books?id=ekW_-1B0fb0C.

[308] E. Palomo, J. North, D. Elizondo, R. Luque, and T. Watson. Visualisation of network forensics traffic data with a self-organising map for qualitative features. In *The 2011 International Joint Conference on Neural Networks*. Institute of Electrical & Electronics Engineers (IEEE), jul 2011. doi: 10.1109/ijcnn.2011.6033434. URL http://dx.doi.org/10.1109/ijcnn.2011.6033434.

[309] S. Park and D. P. O'Leary. Portfolio selection using tikhonov filtering to estimate the covariance matrix. *SIAM Journal on Financial Mathematics*, 1 (1):932–961, jan 2010. doi: 10.1137/090749372. URL http://dx.doi.org/10.1137/090749372.

[310] M. Pergler and A. Freeman. Probabilistic modeling as an exploratory decision-making tool. In *McKinsey Working Paper on Risk*, volume 6. 2008.

[311] B. Pfaff. PSPP. https://www.gnu.org/software/pspp/. accessed: 10.05.2015.

[312] A. Piegat. *Fuzzy Modeling and Control*. Studies in Fuzziness and Soft Computing. Physica-Verlag HD, 2001. ISBN 9783790813852. URL http://books.google.no/books?id=329oSfh-vxsC.

[313] D. W. Pitz and J. W. Shavlik. Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge-Based Systems*, 8 (6):301 – 311, 1995. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/0950-7051(96)81915-0. Knowledge-based neural networks.

[314] V. Plagianakos, D. Sotiropoulos, and M. Vrahatis. Automatic adaptation of learning rate for backpropagation neural networks. *Recent Advances in Circuits and Systems*, (337), 2014.

[315] J. Planquart. Application of neural networks to intrusion detection. SANS Institute, 2001.

[316] A. Plonk and A. Carblanc. Malicious software (malware): A security threat to the internet economy. 2008.

[317] S. Posel. Facebook designing AI 'thought police' to monitor posts. http://www.occupycorporatism.com/home/facebook-

designing-ai-thought-police-monitor-posts/, December 2014. accessed: 12.12.2014.

[318] A. Prakash, E. Venkataramani, H. Yin, and Z. Lin. On the trustworthiness of memory analysis - an empirical study from the perspective of binary execution. *IEEE Transactions on Dependable and Secure Computing*, 12 (5):557–570, Sept 2015. ISSN 1545-5971.

[319] S. Prandl, M. Lazarescu, and D.-S. Pham. A study of web application firewall solutions. In *Information Systems Security*, pages 501–510. Springer, 2015.

[320] M. Pratama, J. Lu, E. Lughofer, G. Zhang, and M. J. Er. Incremental learning of concept drift using evolving type-2 recurrent fuzzy neural network. *IEEE Transactions on Fuzzy Systems*, PP(99):1–1, 2016. ISSN 1063-6706. doi: 10.1109/TFUZZ.2016.2599855.

[321] W. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007. ISBN 9780521880688.

[322] E. Principle. Extension principle. http://equipe.nce.ufrj.br/adriano/fuzzy/transparencias/ExtensionPrinciple/extensionprinciplehandout.pdf, September 2011. access: 18.09.2014.

[323] V. Prokhorenko, K.-K. R. Choo, and H. Ashman. Web application protection techniques: A taxonomy. *Journal of Network and Computer Applications*, 60:95–112, 2016.

[324] M. Pugh, J. Brewer, and J. Kvam. Sensor fusion for intrusion detection under false alarm constraints. In *2015 IEEE Sensors Applications Symposium (SAS)*, pages 1–6, April 2015. doi: 10.1109/SAS.2015.7133634.

[325] R. Puzyriov. Digital forensics tool testing. Master thesis, Gjøvik University College, 2013.

[326] S. Qi, M. Xu, and N. Zheng. A malware variant detection method based on byte randomness test. *Journal of Computers*, 8(10):2469–2477, 2013.

[327] F. Qiu. Neuro-fuzzy based analysis of hyperspectral imagery. volume 74, pages 1235–1247. American Society for Photogrammetry and Remote Sensing, oct 2008. doi: 10.14358/pers.74.10.1235. URL http://dx.doi.org/10.14358/pers.74.10.1235.

[328] F. Qiu. Hyperspectral image classification using an unsupervised neuro-fuzzy system. *Journal of Applied Remote Sensing*, 6(1):063515, apr 2012. doi: 10.1117/1.jrs.6.063515. URL http://dx.doi.org/10.1117/1.jrs.6.063515.

[329] D. Quick and K.-K. R. Choo. Impacts of increasing volume of digital forensic data: A survey and future research challenges. *Digital Investigation*, 11(4):273 – 294, 2014. ISSN 1742-2876. doi: http://dx.doi.org/10.1016/j.diin.2014.09.002. URL http://www.sciencedirect.com/science/article/pii/S1742287614001066.

[330] K. S. M. Raja and K. J. Kumar. Diversified intrusion detection using various detection methodologies with sensor fusion. In *2014 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, pages 442–448, April 2014. doi: 10.1109/ICCPEIC.2014.6915405.

[331] S. Ravi, N. Balakrishnan, and B. Venkatesh. Behavior-based malware analysis using profile hidden markov models. In *Security and Cryptography (SECRYPT), 2013 International Conference on*, pages 1–12, July 2013.

[332] D. K. S. Reddy and A. K. Pujari. N-gram analysis for computer virus detection. *Journal in Computer Virology*, 2(3):231–239, 2006.

[333] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and classification of malware behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-70541-3.

[334] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19 (4):639–668, 2011.

[335] I. Rish. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. IBM New York, 2001.

[336] R. Rojas and J. Feldman. *Neural Networks: A Systematic Introduction*. Springer Berlin Heidelberg, 2013. ISBN 9783642610684.

[337] H. J. Rommelfanger. Fuzzy logic based decision support systems. In *EUSFLAT Conf.*, pages 305–307, 2001.

[338] D. Roobaert, G. Karakoulas, and N. V. Chawla. Information gain, correlation and support vector machines. In *Feature Extraction*, pages 463–470. Springer, 2006.

[339] T. Ross. *Fuzzy Logic with Engineering Applications*. Wiley, 2009. ISBN 9780470748510. URL http://books.google.no/books?id=nhz1f9j6_SMC.

[340] M. Rostamipour and B. Sadeghiyan. Network attack origin forensics with fuzzy logic. In *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*. Institute of Electrical & Electronics Engineers (IEEE), oct 2015. doi: 10.1109/iccke.2015.7365863. URL http://dx.doi.org/10.1109/iccke.2015.7365863.

[341] R. Rowlingson. A ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3):1–28, 2004.

[342] S. Roy. Factors influencing the choice of a learning rate for a backpropagation neural network. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 1, pages 503–507 vol.1, Jun 1994.

[343] S. Rughooputh and H. Rughooputh. Forensic application of a novel hybrid neural network. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 5, pages 3143–3146 vol.5, 1999. doi: 10.1109/IJCNN.1999.836154.

[344] D. Saad and S. A. Solla. On-line learning in multilayer neural networks. In *Mathematics of Neural Networks*, pages 306–311. Springer, 1997.

[345] E. Sahafizadeh and S. Parsa. Survey on access control models. In *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, volume 1, pages V1–1–V1–3, May 2010.

[346] J. Sahs and L. Khan. A machine learning approach to Android malware detection. In *Intelligence and security informatics conference (eisic), 2012 european*, pages 141–147. IEEE, 2012.

[347] K. S. Sajan, B. Tyagi, and V. Kumar. Genetic algorithm based artificial neural network model for voltage stability monitoring. In *Power Systems Conference (NPSC), 2014 Eighteenth National*, pages 1–5, Dec 2014.

[348] S. Saleem. *Protecting the Integrity of Digital Evidence and Basic Human Rights During the Process of Digital Forensics*. PhD thesis, Department of Computer and Systems Sciences, Stockholm University, 2015.

[349] S. Saleem, O. Popov, and I. Bagilli. Extended abstract digital forensics model with preservation and protection as umbrella principles. *Procedia Computer Science*, 35:812 – 821, 2014. ISSN 1877-0509. doi: http://dx.doi.org/10.1016/j.procs.2014.08.246. URL http://www.sciencedirect.com/science/article/pii/S1877050914012113.

[350] Z. Salek, F. M. Madani, and R. Azmi. Intrusion detection using neuarl networks trained by differential evaluation algorithm. In *Information Security and Cryptology (ISCISC), 2013 10th International ISC Conference on*, pages 1–6, Aug 2013. doi: 10.1109/ISCISC.2013.6767341.

[351] R. Salomon. The curse of high-dimensional search spaces: observing premature convergence in unimodal functions. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 918–923 Vol.1, June 2004.

[352] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze. Malware detection based on mining API calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1020–1025, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. doi: 10.1145/1774088.1774303. URL http://doi.acm.org/10.1145/1774088.1774303.

[353] I. Santos, X. Ugarte-Pedrero, B. Sanz, C. Laorden, and P. G. Bringas. Collective classification for packed executable identification. In *Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*, CEAS '11, pages 23–30, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0788-8. doi: 10.1145/2030376.2030379. URL http://doi.acm.org/10.1145/2030376.2030379.

[354] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231:64–82, 2013.

[355] P. Sarlin and T. Eklund. Fuzzy clustering of the self-organizing map: Some applications on financial time series. In *Proceedings of the 8th International Conference on Advances in Self-organizing Maps*, WSOM'11, pages 40–50, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21565-0.

[356] M. Schiffman. A brief history of malware obfuscation. http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2, 2010. accessed: 13.07.2016.

[357] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[358] C. R. Schmidt. *Effect of irregular topology in shperical Self-Organizing Maps*. PhD thesis, San Diego State University, December 2008.

[359] I. Schmidtmann, G. Hammer, M. Sariyar, A. Gerhold-Ay, and K. des öffentlichen Rechts. Evaluation des krebsregisters NRW schwerpunkt record linkage. *Abschlußbericht vom*, 11, 2009.

[360] G. Schudel. Bandwidth, packets per second, and other network performance metrics. *Abgerufen am*, 10:2010, 2010.

[361] S. Senel-Kleine, J. Bouche, and M. Kappes. On the usefulness of machine learning techniques in collaborative anomaly detection. In *Internet Technologies and Applications (ITA), 2015*, pages 213–218, Sept 2015.

[362] M. Z. Shafiq, M. Farooq, and S. A. Khayam. A comparative study of fuzzy inference systems, neural networks and adaptive neuro fuzzy inference systems for portscan detection. In *Applications of Evolutionary Computing*, pages 52–61. Springer, 2008.

[363] K. Shah, N. Dave, and S. Chavon. Adaptive neuro-fuzzy intrusion detection system. Proceeding IEEE International Conference Information Technology: Coding and Computing, 2004.

[364] S. Shah. Top ten web attacks. White paper, 2002.

[365] H. Shahriar, M. Islam, and V. Clincy. Android malware detection using permission analysis. In *SoutheastCon 2017*, pages 1–6, March 2017. doi: 10.1109/SECON.2017.7925347.

[366] R. Shahzad, N. Lavesson, and H. Johnson. Accurate adware detection using opcode sequence extraction. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 189–195, Aug 2011. doi: 10.1109/ARES.2011.35.

[367] A. Shalaginov. Soft computing and hybrid intelligence for decision support in forensics science. In *IEEE Intelligence and Security Informatics 2016*, pages 304–309, 2016.

[368] A. Shalaginov. Dynamic feature-based expansion of fuzzy sets in neuro-fuzzy for proactive malware detection. In *Information Fusion (Fusion), 2017 20th International Conference on*, pages 1–8. IEEE, 2017.

[369] A. Shalaginov. Fuzzy logic model for digital forensics: A trade-off between accuracy, complexity and interpretability. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[370] A. Shalaginov. Evolutionary optimization of on-line multilayer perceptron for similarity-based access control. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 823–830, May 2017. doi: 10.1109/IJCNN.2017.7965937.

[371] A. Shalaginov and K. Franke. Automatic rule-mining for malware detection employing neuro-fuzzy approach. *Norsk informasjonssikkerhetskonferanse (NISK)*, 2013.

[372] A. Shalaginov and K. Franke. Towards improvement of multinomial classification accuracy of neuro-fuzzy for digital forensics applications. In *15th International Conference on Hybrid Intelligent Systems (HIS 2015)*, volume 420, pages 199–210. Springer Publishing, 2015.

[373] A. Shalaginov and K. Franke. A new method of fuzzy patches construction in neuro-fuzzy for malware detection. In *Conference of the International Fuzzy Systems Association and European Society for Fuzzy Logic and Technology (IFSA-EUSFLAT)*. Atlantis Press, 2015.

[374] A. Shalaginov and K. Franke. A new method for an optimal som size determination in neuro-fuzzy for the digital forensics applications. In *International Work-Conference on Artificial Neural Networks*, pages 549–563. Springer International Publishing, 2015.

[375] A. Shalaginov and K. Franke. Automated generation of fuzzy rules from large-scale network traffic analysis in digital forensics investigations. In *7th International Conference on Soft Computing and Pattern Recognition (SoCPaR 2015)*. IEEE, 2015.

[376] A. Shalaginov and K. Franke. Big data analytics by automated generation of fuzzy rules for network forensics readiness. *Applied Soft Computing*, 2016.

[377] A. Shalaginov and K. Franke. Multinomial classification of web attacks using improved fuzzy rules learning by neuro-fuzzy. *International Journal of Hybrid Intelligent Systems*, 13(1):15–26, 2016.

[378] A. Shalaginov and K. Franke. Intelligent generation of fuzzy rules for network firewalls based on the analysis of large-scale network traffic dumps. *International Journal of Hybrid Intelligent Systems*, 13(3-4):195–206, 2016.

[379] A. Shalaginov and K. Franke. Automated intelligent multinomial classification of malware species using dynamic behavioural analysis. In *2016 14th*

Annual Conference on Privacy, Security and Trust (PST), pages 70–77, Dec 2016. doi: 10.1109/PST.2016.7906939.

[380] A. Shalaginov and K. Franke. A deep neuro-fuzzy method for multi-label Windows PE32 malware classification. *IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2017)*, 2017. (Submitted).

[381] A. Shalaginov, L. S. Grini, and K. Franke. Understanding neuro-fuzzy on a class of multinomial malware detection problems. In *IEEE International Joint Conference on Neural Networks (IJCNN) 2016*, pages 684–691. Research Publishing Services, 2016.

[382] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke. Machine learning aided static malware analysis: A survey and tutorial. *Cyber Threat Intelligence 2017*, 2017.

[383] C. R. Shalizi. Advanced data analysis from elementary point of view. Technical report, Undergraduate Advanced Data Analysis, Department of Statistics, Carnegie Mellon University, 2013.

[384] K. Shang and Z. Hossen. Applying fuzzy logic to risk assessment and decision-making. Technical report, CAS/CIA/SOA Join Risk Management Section, 2013.

[385] M. Shankarapani, K. Kancherla, S. Ramammoorthy, R. Movva, and S. Mukkamala. Kernel machines for malware classification and similarity analysis. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–6. IEEE, 2010.

[386] R. Shanmugavadivu and N. Nagarajan. Network intrusion detection system using fuzzy logic. *Indian Journal of computer Science and Engineering*, 2011.

[387] A. F. Shapiro and M.-C. Koissi. Risk assessment applications of fuzzy logic. Technical report, Canadian Institute of Actuaries, 2015.

[388] N. M. Share. Desktop operating system market share. http://www.netmarketshare.com/report.aspx?qprid=10&qptimeframe=M&qpsp=200&qpch=350&qpcustomd=0, September 2015. accessed: 15.10.2015.

[389] S. Shiaeles. *Real time detection and response of distributed denial of service attacks for web services*. PhD thesis, Democritus University of Thrace, 2013.

[390] D. Shinder. The pros and cons of behavioral based, signature based and whitelist based security. http://www.windowsecurity.com/articles-tutorials/misc_network_security/Pros-Cons-Behavioral-Signature-Whitelist-Security.html, November 2008. Accessed: 25.07.2016.

[391] R. Sindal and S. Tokekar. Adaptive soft handoff based neuro-fuzzy call admission control scheme for multiclass calls in cdma cellular network. In *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on*, pages 279–284, March 2012.

[392] H. Singh. Performance analysis of unsupervised machine learning techniques for network traffic classification. In *Advanced Computing Communication Technologies (ACCT), 2015 Fifth International Conference on*, pages 401–404, Feb 2015. doi: 10.1109/ACCT.2015.54.

[393] J. Singh and M. J. Nene. A survey on machine learning techniques for intrusion detection systems. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(11), 2013.

[394] R. Singh, H. Kumar, and R. Singla. Review of soft computing in malware detection. *Special issues on IP Multimedia Communications*, 1(1):55–60, October 2011. Full text available.

[395] W. Siripanwattana and S. Srinoy. Information security based on soft computing techniques, 2008.

[396] L. I. Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 26 2002. URL http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf.

[397] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4): 427 – 437, 2009. ISSN 0306-4573.

[398] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[399] Sophos. When malware goes mobile. https://www.sophos.com/en-us/security-news-trends/security-trends/malware-goes-mobile.aspx. accessed: 03.05.2017.

[400] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1453–1460. IEEE, 2011.

[401] A. Stein. Properties of limits. http://www.math.uconn.edu/~stein/math115/slides/math115-130notes.pdf.

[402] K. Stoffel, P. Cotofrei, and D. Han. Fuzzy methods for forensic data analysis. In *SoCPaR*, pages 23–28, 2010.

[403] D. Sule. Importance of forensic readiness. ISACA journal, https://www.isaca.org/Journal/archives/2014/Volume-1/Pages/JOnline-Importance-of-Forensic-Readiness.aspx, 2014. accessed: 18.05.2017.

[404] L. Sun, S. Versteeg, S. Boztaş, and T. Yann. Pattern recognition techniques for the classification of malware packers. In *Information security and privacy*, pages 370–390. Springer, 2010.

[405] G. Surman. Understanding security using the osi model. *SANS Institute InfoSec Reading Room*, 2002.

[406] SWGDE. Best practices for computer forensics. https://www.swgde.org/documents/Current%20Documents/SWGDE%20Best%20Practices%20for%20Computer%20Forensics, September 2014. accessed: 03.02.2017.

[407] J. Székely, M. L. Rizzo, and N. K. Bakirov. Measuring and testing dependence by correlation of distances.

[408] S. Tahilyani, M. Darbari, and P. K. Shukla. Soft computing approaches in traffic control systems: A review. *{AASRI} Procedia*, 4(0):206 – 211, 2013. ISSN 2212-6716. doi: http://dx.doi.org/10.1016/j.aasri.2013.10.032. URL http://www.sciencedirect.com/science/article/pii/S2212671613000334. 2013 {AASRI} Conference on Intelligent Systems and Control.

[409] M. Takács. Parameters and rules of fuzzy-based risk management models. 2011.

[410] S. Tang. The detection of trojan horse based on the data mining. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*, volume 1, pages 311–314, Aug 2009. doi: 10.1109/FSKD.2009.354.

[411] S. Tano, T. Oyama, and T. Arnould. Deep combination of fuzzy inference and neural network in fuzzy inference software—finest. *Fuzzy Sets and Systems*, 82(2):151–160, 1996.

[412] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[413] C. Thang, P. Q. Toan, E. Cooper, and K. Kamei. Application of soft computing to tax fraud detection in small businesses. In *Communications and Electronics, 2006. ICCE '06. First International Conference on*, pages 402–407, Oct 2006. doi: 10.1109/CCE.2006.350887.

[414] G. Thimm and E. Fiesler. High-order and multilayer perceptron initialization. *Neural Networks, IEEE Transactions on*, 8(2):349–359, 1997.

[415] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[416] V. Torra. Trends in information fusion in data mining. In *Information fusion in data mining*, pages 1–6. Springer, 2003.

[417] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera. MRPR: A MapReduce solution for prototype reduction in big data classification. *Neurocomputing*, 150:331–345, feb 2015. doi: 10.1016/j.neucom.2014.04.078. URL http://dx.doi.org/10.1016/j.neucom.2014.04.078.

[418] J. V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225 – 1231, 1996. ISSN 0895-4356. doi: http://dx.doi.org/10.1016/S0895-4356(96)00002-9.

[419] B. u. Islam, Z. Baharudin, M. Q. Raza, and P. Nallagownden. Optimization of neural network architecture using genetic algorithm for load forecasting. In *Intelligent and Advanced Systems (ICIAS), 2014 5th International Conference on*, pages 1–6, June 2014.

[420] X. Ugarte-Pedrero, I. Santos, P. Bringas, M. Gastesi, and J. Esparza. Semi-supervised learning for packed executable detection. In *Network and System Security (NSS), 2011 5th International Conference on*, pages 342–346, Sept 2011. doi: 10.1109/ICNSS.2011.6060027.

[421] R. Unuchek. Mobile malware evolution 2016. https://securelist.com/analysis/kaspersky-security-bulletin/77681/mobile-malware-evolution-2016/, February 2017. accessed: 03.05.2017.

[422] D. Uppal, R. Sinha, V. Mehra, and V. Jain. Malware detection and classification based on extraction of API sequences. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 2337–2342. IEEE, 2014.

[423] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE transactions on neural networks*, 11(3):586–600, 2000.

[424] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas. Self-organizing map in matlab: the SOM Toolbox. In *In Proceedings of the Matlab DSP Conference*, pages 35–40, 2000.

[425] J. Vieira, F. M. Dias, and A. Mota. Neuro-fuzzy systems: a survey. In *5th WSEAS NNA International Conference*, 2004.

[426] J. Vincent. 99.6 percent of new smartphones run Android or iOS. https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016, February 2017. accessed: 19.05.2017.

[427] C. Visual and B. Unit. Microsoft Portable Executable and common object file format specification, 1999.

[428] I. Vural and H. Venter. Using network forensics and artificial intelligence techniques to detect bot-nets on an organizational network. In *2010 Seventh International Conference on Information Technology: New Generations*. Institute of Electrical & Electronics Engineers (IEEE), 2010. doi: 10.1109/itng.2010.67. URL http://dx.doi.org/10.1109/itng.2010.67.

[429] C. Wang, Z. Qin, J. Zhang, and H. Yin. A malware variants detection methodology with an opcode based feature method and a fast density based clustering algorithm. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 481–487, Aug 2016. doi: 10.1109/FSKD.2016.7603221.

[430] N. Wang, D. Wang, and Z. Wu. An online self-organizing neuro-fuzzy system from training data. In *Advanced Computational Intelligence (IWACI), 2010 Third International Workshop on*, pages 26–31, Aug 2010. doi: 10.1109/IWACI.2010.5585231.

[431] T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh. Detecting unknown malicious executables using portable executable headers. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 278–284, Aug 2009. doi: 10.1109/NCM.2009.385.

[432] G. Wangen and A. Shalaginov. Quantitative risk, statistical methods and the four quadrants for information security. In *The 10th International Conference on Risks and Security of Internet and Systems, 2015. CRiSIS'15*. Springer, 2015.

[433] G. Wangen, A. Shalaginov, and C. Hallstensen. Cyber security risk assessment of a DDoS attack. In *International Conference on Information Security*, pages 183–202. Springer International Publishing, 2016.

[434] E. W. Weisstein. Central limit theorem. http://mathworld.wolfram.com/CentralLimitTheorem.html. accessed: 20.02.2017.

[435] J. Weizenbaum. *Computer Power and Human Reason: From Judgment to Calculation*. W. H. Freeman & Co., New York, NY, USA, 1976. ISBN 0716704641.

[436] R. Wentworth. Computer virus! http://uanr.com/articles/virus.html. accessed: 09.02.2016.

[437] S. Whiteson and D. Whiteson. Machine learning for event selection in high energy physics. *Engineering Applications of Artificial Intelligence*, 22(8): 1203–1217, dec 2009. doi: 10.1016/j.engappai.2009.05.004. URL http://dx.doi.org/10.1016/j.engappai.2009.05.004.

[438] B. Widrow and M. A. Lehr. The handbook of brain theory and neural networks. chapter Perceptrons, Adalines, and Backpropagation, pages 719–724. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-51102-9.

[439] Wikipedia. Timeline of computer viruses and worms. https://en.wikipedia.org/wiki/Timeline_of_computer_viruses_and_worms, . accessed: 09.02.2016.

[440] Wikipedia. Timeline of Microsoft Windows. https://en.wikipedia.org/wiki/Timeline_of_Microsoft_Windows, . accessed: 08.02.2016.

[441] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[442] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[443] S. Woodall. Firewall design principles. *Computer Networks and Computer Security. Coursework paper, North Carolina State University, USA*, 2004.

[444] C. Wu and J. Irwin. *Introduction to Computer Networks and Cybersecurity*. CRC Press, 2016. ISBN 9781466572140.

[445] D. Yadron, S. Ackerman, and S. Thielman. Inside the fbi's encryption battle with Apple. *Guardian*, 2016.

[446] Y. Yamamoto and K. Iwanuma. Online pattern mining for high-dimensional data streams. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2880–2882, Oct 2015.

[447] X. Yan. Study on the dynamic forensics method based on BP neural network. In *2011 International Conference on Future Computer Sciences and Application*. Institute of Electrical & Electronics Engineers (IEEE), jun 2011. doi: 10.1109/icfcsa.2011.58. URL http://dx.doi.org/10.1109/icfcsa.2011.58.

[448] X.-S. Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[449] Y. Ye, D. Wang, T. Li, and D. Ye. IMDS: intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 1043–1047, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-609-7.

[450] S. Yu, G. Gu, A. Barnawi, S. Guo, and I. Stojmenovic. Malware propagation in large-scale networks. *IEEE Transactions on Knowledge & Data Engineering*, (1):170–179, 2015.

[451] X.-H. Yu and G.-A. Chen. Efficient backpropagation learning using optimal learning rate and momentum. *Neural Netw.*, 10(3):517–527, Apr. 1997. ISSN 0893-6080.

[452] M. Zabidi, M. Maarof, and A. Zainal. Malware analysis with multiple features. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 231–235, March 2012. doi: 10.1109/UKSim.2012.40.

[453] L. A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[454] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy sets and systems*, 1(1):3–28, 1978.

[455] L. A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, Mar. 1994. ISSN 0001-0782. doi: 10.1145/175247.175255. URL http://doi.acm.org/10.1145/175247.175255.

[456] M. Zamani. Machine learning techniques for intrusion detection. 2013.

[457] L. Zeltser. Tools for analyzing static properties of suspicious files on Windows. http://digital-forensics.sans.org/blog/2014/03/04/tools-for-analyzing-static-properties-of-suspicious-files-on-windows. accessed: 09.02.2016.

[458] B. Zhang, J. Yin, J. Hao, D. Zhang, and S. Wang. Malicious codes detection based on ensemble learning. In *Proceedings of the 4th International Conference on Autonomic and Trusted Computing*, ATC'07, pages 468–477, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-73546-1, 978-3-540-73546-5.

[459] Y. Zhang, editor. *Machine Learning*. INFTECH, isbn 978-953-307-033-9, edition, February 2010.

[460] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi. Real-time network anomaly detection system using machine learning. In *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the*, pages 267–270, March 2015.

[461] Z. Zhao. A virus detection scheme based on features of control flow graph. In *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, pages 943–947, Aug 2011. doi: 10.1109/AIMSEC.2011.6010676.

[462] S. Zhou, Q. Chen, and X. Wang. Fuzzy deep belief networks for semi-supervised sentiment classification. *Neurocomputing*, 131:312–322, 2014.

[463] H. Zimmermann. Osi reference model–the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28 (4):425–432, 1980.

Appendix A

Computational Setup & Used Hardware

This work was done to improve existing Neuro-Fuzzy that also required comprehensive experimental evaluation and implementation of various functionality for proof of concept demonstration. To be able to fulfil this we in timely matter I used Dedicated Server available at NTNU Digital Forensics and also implemented a number to tools. The details are given below.

A.1 Developed Software

Novel theoretical contributions required designing and development of corresponding software tools that should be capable of demonstrating not only newly proposed improvements of Neuro-Fuzzy, but also comparison to original methods, including Kosko Fuzzy Inference System. In addition, this thesis investigates analysis of large-scale data in a timely manner, meaning that we had to perform rigorous mathematical and parallel optimization. Below it will be given an outline of the specific software tools and also report by CLOC[1] to show the quantitative statistics of the code.

A.1.1 Implementation of Neuro-Fuzzy Method and Self-Organizing Map Library

Improvement of the Neuro-Fuzzy method for Digital Forensics Investigation required proof-of-concept demonstration to be able to highlight not only theoretical enhancement, yet also show the practical benefits when it comes to complex and large-scale data. I implemented all the steps described in the Chapter 3. In ad-

[1] http://cloc.sourceforge.net/

dition to this it was implemented functionality also for Simple Rectangular and Kosko Fuzzy Inference Systems as was reviewed in the Chapter 2. The method consisted of two stages. The first part was based on the designed and implemented Self-Organizing Map library using C++ and STL, Boost, Eigen, OpenMP libraries in order to be able to process large-scale data in a fast matter using optimized matrix operations on several execution threads. Chapter 4 showed different aspects and time complexity of the implemented method. Second stage was based on the ANN learning, where automatically extracted fuzzy rules where used to train and tune the classification model. It should be mentioned that implemented library is fast and verified during many experiments, also scalable and does not depend on number of features, classes, data site or number of execution threads on the processor within the hardware capabilities.

A.1.2 Processing of PE32 malware files and VirusTotal response

The first part of this work was devoted to implementation of automated engine that will be processing PE32 malware files and store all necessary information (features, anti-virus reports, etc.). PEframe[2] and VirusTotal[3] were chosen to be sources of the relevant characteristic of the binary files. Virus Total in addition to PE32 header fields provide reports from about 60 anti-virus vendors. PHP and MySQL were used to retrieve all relevant information using VirusTotal Private API. Files preprocessing was performed using Linux native bash, while analysis of the reports in JSON were done with PHP and Python.

Second part of the experiments included development of the automated procedure for malware analysis in a sandbox. Sandbox and files rotation was done with a help of bash and VirtualBox API, while parts written in PHP were used to analyse the data and derives corresponding numerical features from CaptureBAT and WinDUMP reports to store them in MySQL.

A.2 Experimental Setups & Used Computing Environments

This chapter contains practical aspects of performed experiments and details of proof of concept demonstration for each of the experimental setup mentions earlier. These guidelines may help to reproduce the flow of the data analysis and preprocessing. The computations have been carried out on (i) 4-threaded VDS with Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz with 4 cores (8 threads), 16GB RAM and SSD RAID storage and installed OS Ubuntu 14.04 64 bit.

1. The software in [377] was implemented in C++ and compiled by *gcc-4.8.2*.

[2] https://github.com/guelfoweb/peframe
[3] https://www.virustotal.com/

To perform a part of mathematical operations, we utilized *Boost* and *Eigen* libraries. Each experiments included: SOM grouping, extracting three type of fuzzy regions parameters (rectangular, Kosko, proposed method) and models training. To estimate the speed of the execution, relative times of the experiments were measured for all samples in training set.

2. To show how the malware analysis can be done automatically using different ML techniques in the contribution [382], we perform following steps for data preprocessing and experimental design. Also we described how we tackled different challenges experienced during experiments. All the experiments were performed on a Virtual Dedicated Server. Files pre-processing were performed using *bash* scripts due to native support in Linux OS. To store extracted features we are going to use MySQL 5.5 database engine together with Python and PH connectors. So, following steps are employed:

   (a) Unprocessed malware and benign files were placed into two directories "malware/" and "benign/".

   (b) To eliminate duplicates, we renamed all the files to their MD5 sums.

   (c) PE32 files were selected in the folder, which was done by executing Linux *file* command:

```
\$ file 000000b4dccfbaa5bd981af2c1bbf59a
000000b4dccfbaa5bd981af2c1bbf59a: PE32 executable (DLL)
    (GUI) Intel 80386, for MS Windows
```

   (d) We scrapped all the files from current directory that and move it to a dedicated one:

```
#!/bin/sh
cd ../windows1;
counter=0;
for i in *; do
     counter=$((counter+1));
     echo "$counter";
     VAR="file␣$i␣|␣grep␣PE32␣" ;
          VAR1=$(eval "$VAR") ;
          len1=${#VAR1};
     if [ -n "$VAR1" ] && [ "$len1" -gt "1" ] ;
     then
          echo "$VAR1" | awk '{print $1}' | awk '{gsub(/:
              $/,""); print $1 "␣../windows/PE/" $1}'|
              xargs mv -f  ;
     else
          echo "other";
              file $i | awk '{print $1}' | awk '{gsub(/:$
                  /,""); print $1 "␣../windows/other/" $1}'
                   | xargs mv -f ;
```

```
16          fi
17 done
```

3. Practical experiments and computations on the Network Forensics Readiness [376] have been carried on the Virtual Machine The implementation of NF was complited using C++ 11 standard with Boost libraries version 1.55, OpenMP libraries version 3.1 and Eigen libraries version 3.2.0-8. The functionality, including SOM grouping, NF training and performance estimation, were implemented in C++ and compiled by gcc version 4.8.2 with parameters *"-m 64 -std=c++11 -fopenmp -DEIGEN_NO_DEBUG"*. Each experiment included: SOM grouping, extracting three types of fuzzy patches parameters (Simple method, Kosko method, Proposed method), learning three models and performance estimation for each sample using defined three models. During the experiments the dataset were first loaded into contingent containers into RAM by the program and then analyzed. No transformation on the original data was done. The performance metrics were collected using double-precision floating-point data structures, so the errors from overflowing the computational grid are neglected at this point. The size of the datasets were: 8.1GB for HIGGS 1, 2.4GB for SUSY 2, 0.2GB for Record Linkage 3, 0.15GB and 0.04GB for full and 10% datasets of KDD CUP 1999 challenge 4 respectively. The files were read by $fread()$ function and put into $std :: vector < boost :: numeric :: ublas :: vector < double >>$ containers for easier further processing.

4. All the experiments on mutlinomial malware analysis in [167] were performed on a Virtual Server. Files pre-processing were performed using *bash* scripts due to native support in Linux OS. To store extracted features MySQL 5.5.46 database engine was used. Characteristics harvesting and feature extraction was implemented in PHP 5.5.9 and run for many days without time limitation. Further, pre-processing, feature selection and classification was performed using opensource *Weka* v 3.7.13 package [149] that contains community-accepted ML methods.

5. All the experiments in the contribution [381] were performed on a Virtual Server. Files pre-processing were performed using *bash* scripts due to native support in Linux OS. However, some tricks were used to handle large datasets with $> 100k$ files such that $ls\ -A \mid wc\ -l$ for fast counting number of files. To store extracted features MySQL 5.5.46 database engine was used. Characteristics harvesting and feature extraction was implemented in PHP 5.5.9 and run for many days without script time limitation.

6. Our experiments for dynamic PE32 malware analysis in [379] were divided

into two parts: (i) acquisition of behavioural characteristics from execution in a controlled environment and (ii) raw log files processing to extract numerical features.

(i) The experiment was performed using *bash* scripts due to native support in Linux OS. The automation was done on the Virtual Box 5.0.20 with Windows 7 32 bits to collect dynamic characteristics. According to studies, this version is still present on 49.05 % of computers in the world, 'being the most widely installed OS [34]. As a handbook we referred partially to the SANS tutorial by Kramer [234] that describes deploying Windows 7 with corresponding tools for malware analysis. Disk activities and processes in memory were monitored using pre-installed CaptureBat 2.0.0-5574 [24] that is also able to dump all modified and deleted files. The tool comes as a part of the The Honeypot Project and designed to allow malware analyst to investigate different activities that are performed during malware execution. Furthermore, network packets were acquired through CaptureBat inside VM with help of WinDump 3.9.5 (tcpdump for MS Windows) [22], which stores all raw network traffic into a pcap file. This however needs to be further processed by tshark 1.10.6 [15] and *capinfos* v 2.1.1 into readable format.

(ii) The raw logs, network traffic and corresponding collected data were precessed. To store extracted features, we are going to use MySQL 5.5.49 database engine together with and PHP 5.5.9 connectors. Weka 3.7.13 [149] was used to evaluate the performance of community-accepted ML methods.

7. All the experiments for Deep Neuro-Fuzzy in [380] were performed on the Virtual Dedicated Server. The following tasks were executed consequently: (1) execution and processing of malware files collection inside a Windows PE32 virtual testing laboratory environment. The automation was done on the Virtual Box 5.0.20 with Windows 7 32bits to collect *dynamic* characteristics as recommended in the SANS tutorial by Kramer [234]. Disk activities and processes in memory were monitored using pre-installed CaptureBat 2.0.0-5574 [24], while network packets were acquired through CaptureBat inside VM with help of WinDump 3.9.5 (tcpdump for MS Windows) [22], which stores all raw network traffic into a pcap file. This however needs to be further processed by tshark 1.10.6 [15] and *capinfos* v 2.1.1 into a readable format. In addition to this, *static* characteristics were retrieved by PEframe [57] 5.0 as well as from VirusTotal collection [18]. (2) Learning and testing classification performance of NF, ANN ad Deep NF. Weka 3.7.13 [149] and RapidMiner [14] 7.2.003 were used to evaluate the performance of community-accepted ML methods. Additionally, we implemented the proposed methodology. Plots were made with a help of statistical

environment R [2] 3.3.2 / RapidMiner.

Appendix B

Empirical Study of the Neuro-Fuzzy Method

During our experiments with Vesanto data have not used bootstrap aggregation, because according to observation it reduces the execution time only on 10-20%. It takes 27 days with parallel optimization to train the proposed method for the SUSY dataset, which is enormous amount of time. It should be also noted that we were not able to get the results for the HIGGS dataset after running the original method over 6 months on the Virtual Server. To contrary, suggested improvements allowed to use no more than couple hours to execute all required experiments on all datasets without reduction in the accuracy, which is one of the major achievements. Finally, inference of the fuzzy rules took around 1-10 milliseconds for all datasets, which is a considerable delay in comparison to the suggested method.

Further, am pbstacle arose when we processed such large datasets. To deal with ill-posed problem that appear in covariance matrix and inverse covariance matrix calculations the Tikhonov regularization was employed as suggested by Park et al. [309]:

$$Cov = Cov + diag \tag{B.1}$$

where each element of diagonal matrix is a small offset $diag_i = 10^{-6}$.

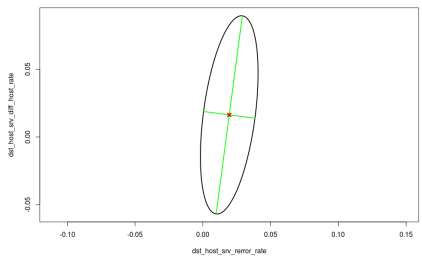
B.1 Example of Derived Fuzzy Rules using Proposed Method

The log of extracted fuzzy rules from KDD CUP 1999 10% dataset using proposed method is given in Figure B.1. It contains numerical measures for each centroid, inverse covariance matrix used in Equation 3.17.

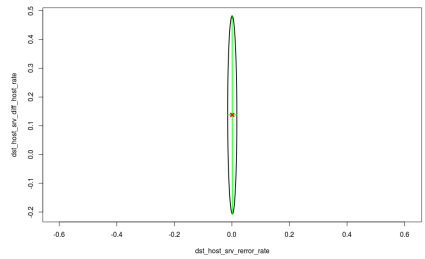
```
Rule number 0
Centroids:
   270.654    12515.7 0.00628931   0.993711          0          0   0.996855  0.0194969  0.0162893
InversedCovariance Matrix:
   0.0001791 -1.81306e-08    0.0196433   -0.0421157            0            0     0.552788     0.102306  -0.00876423
-1.81306e-08  1.72832e-09 -1.59243e-05 -7.47808e-06            0            0  5.42458e-05 -0.000551823  6.55002e-05
   0.0196433 -1.59243e-05      163.222      -6.5043            0            0       50.481      65.9089     -4.83962
  -0.0421157 -7.47808e-06      -6.5043      170.938            0            0     -121.481     -70.6926      4.97834
           0            0            0            0       1e+06            0            0            0            0
           0            0            0            0            0       1e+06            0            0            0
    0.552788  5.42458e-05       50.481     -121.481           -0           -0      2757.96     -225.954      224.475
    0.102306 -0.000551823      65.9089     -70.6926            0            0     -225.954      2910.33     -282.757
 -0.00876423  6.55002e-05     -4.83962      4.97834           -0           -0      224.475     -282.757       155.28
Class ID:1

-------------------------------------------------------------------------------------------------------------------
     Rule number 34
Centroids:
   274.035          0          0 0.00293255          0          0   0.405132 0.00117302    0.13783
InversedCovariance Matrix:
3.25832e-05          0          0 0.000744409          0          0   -0.025841   0.0937003  0.00241228
          0       1e+06          0          0          0          0          0          0          0
          0          0       1e+06          0          0          0          0          0          0
0.000744409          0          0      439.797          0          0    -1.93826    -695.279   0.180938
          0          0          0          0       1e+06          0          0          0          0
          0          0          0          0          0       1e+06          0          0          0
  -0.025841          0          0    -1.93826          0          0    25.7222    -85.4283  -0.299251
  0.0937003          0          0    -695.279          0          0   -85.4283     5328.77     7.9748
 0.00241228          0          0    0.180938          0          0  -0.299251     7.9748     9.0998
Class ID:2
```

Figure B.1: Parameters of the extracted fuzzy rules using proposed method

Previous mentioned elliptic fuzzy rules can be sketched using 2D ellipses as show in Figures B.2. For demonstration purposes we used 8^{th} and 9^{th} attributes in the KDD CUP 1999 dataset, which are *dst_host_srv_diff_host_rate* and *dst_host_srv_rerror_rate* respectively. Also for better visualization we used scaled ellipse.



(a) Rule number 0 extracted from KDD CUP 1999 10% dataset, class 1



(b) Rule number 34 extracted from KDD CUP 1999 10% dataset, class 2

Figure B.2: Visualization of fuzzy rules extracted by Neuro-Fuzzy

B.2 Accuracy of Neuro-Fuzzy with Manually-defined SOM Size

To evaluated the hypothesis about SOM size we performed comparison of cross-validated results with bootstrapped training data set as mentioned earlier. Here we made an assumption that 95% of the samples should contain in each of the fuzzy patches. However, this can be changed automatically depending on the values of spread with respect to central tendency. The SOM size over 10x10 on the 1^{st} stage

of NF was not considered since the amount of rules will be about 100 that makes the model complicated to perceive. The performance was evaluated using the (i) regression performance metrics Mean Absolute Error $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - d_i|$, Mean Absolute Percent Error $MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{y_i - d_i}{d_i}| \cdot 100\%$ and Relative Absolute Error $RAE = \frac{\sum_{i=1}^{N}|y_i - d_i|}{\sum_{i=1}^{N}|d_i - \bar{d}|}$ for the output of the 2^{nd} stage of the Neuro-Fuzzy and (ii) classification accuracy ("Acc, %") of derived rule-based model based on the *min-max* principle. The results for all 8 datasets are presented in the Tables B.1-B.8 .

Table B.1: Performance comparison of the simple rectangular, Kosko and Gaussian on the Climate Model Simulation Crashes dataset

SOM	Eff.rules	Method	Performance			
			MAE	MAPE,%	RAE	Acc ,%
3x3	15	Simple	0.712	37.049	4.485	23.333
		Kosko	0.912	45.640	5.743	91.481
		Gaussian	0.031	3.150	0.1982	**97.037**
5x5	31	Simple	0.457	25.855	2.8814	21.296
		Kosko	0.859	43.232	5.408	91.296
		Gaussian	0.046	4.632	0.291	95.370
10x10	62	Simple	0.210	14.459	1.323	31.111
		Kosko	0.994	50.003	6.257	91.296
		Gaussian	0.053	4.869	0.339	90.185

Table B.2: Performance comparison of the simple rectangular, Kosko and Gaussian on the Fertility dataset

SOM	Eff.rules	MF method	Performance			
			MAE	MAPE,%	RAE	Acc ,%
3x3	9	Simple	0.186	13.031	0.884	88.000
		Kosko	0.120	6.000	0.568	88.000
		Gaussian	0.076	4.323	0.362	88.000
5x5	9	Simple	0.284	22.949	1.345	87.000
		Kosko	0.172	12.179	0.816	78.000
		Gaussian	0.092	5.414	0.437	78.000
10x10	9	Simple	0.232	18.270	1.101	88.000
		Kosko	0.149	9.499	0.706	85.000
		Gaussian	0.101	6.456	0.480	**92.000**

Table B.3: Performance comparison of the simple rectangular, Kosko and Gaussian on the Banknote Authentication dataset

SOM	Eff.rules	MF method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc** ,%
3x3	17	Simple	0.103	5.654	0.209	96.1370
		Kosko	0.446	22.369	0.903	92.638
		Gaussian	0.063	6.284	0.128	**100.000**
5x5	37	Simple	0.045	3.9438	0.092	95.55
		Kosko	0.428	30.727	0.867	96.282
		Gaussian	0.038	3.839	0.078	99.927
10x10	94	Simple	0.438	21.938	0.888	55.539
		Kosko	0.450	22.782	0.911	99.198
		Gaussian	0.042	4.168	0.086	**100.000**

Table B.4: Performance comparison of the simple rectangular, Kosko and Gaussian on the Mobile Malware dataset

SOM	Eff.rules	MF method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc**,%
3x3	18	Simple	0.572	37.616	1.179	41.344
		Kosko	0.407	40.704	0.839	58.487
		Gaussian	0.070	4.973	0.144	**91.764**
5x5	46	Simple	0.599	39.675	1.234	17.647
		Kosko	0.309	28.355	0.637	70.420
		Gaussian	0.094	7.109	0.194	90.588
10x10	61	Simple	0.571	40.103	1.177	40.840
		Kosko	0.292	24.992	0.602	77.815
		Gaussian	0.122	9.226	0.253	86.890

Table B.5: Performance comparison of the simple rectangular, Kosko and Gaussian on the Ionosphere dataset

SOM	Eff.rules	Method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc** ,%
3x3	16	Simple	0.3976	26.0414	0.8639	59.5442
		Kosko	0.5134	47.8038	1.1154	40.7407
		Gaussian	0.1442	11.5710	0.3133	**88.6040**
5x5	23	Simple	0.4653	34.3080	1.0111	45.2991
		Kosko	0.4832	44.4185	1.0499	51.8519
		Gaussian	0.2174	17.9925	0.4724	84.3305
10x10	17	Simple	0.4711	32.9836	1.0237	30.1994
		Kosko	0.2533	12.7915	0.5505	51.2821
		Gaussian	0.3017	21.8703	0.6555	74.3590

Table B.6: Performance comparison of the simple rectangular, Kosko and Gaussian on the SPECTF Heart dataset

SOM	Eff.rules	Method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc** ,%
3x3	4	Simple	0.6164	35.3766	1.2327	45.0000
		Kosko	0.4928	31.3602	0.9857	72.5000
		Gaussian	0.1594	10.3125	0.3187	85.0000
5x5	3	Simple	0.6677	40.7887	1.3353	41.2500
		Kosko	0.4942	33.6323	0.9885	72.5000
		Gaussian	0.1833	13.3333	0.3667	85.0000
10x10	2	Simple	0.9801	67.6425	1.9603	31.2500
		Kosko	0.5000	25.0000	1.0000	75.0000
		Gaussian	0.2000	16.5625	0.4000	**86.2500**

Table B.7: Performance comparison of the simple rectangular, Kosko and Gaussian on the Madelon dataset

SOM	Eff.rules	Method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc** ,%
3x3	18	Simple	0.5078	37.1075	1.0157	49.4500
		Kosko	0.5005	27.1667	1.0009	56.3500
		Gaussian	0.5395	36.1000	1.0790	**82.1500**
5x5	50	Simple	0.5110	39.9980	1.0220	49.3000
		Kosko	0.5000	37.5092	0.9999	57.4000
		Gaussian	0.5395	36.1000	1.0790	**82.1500**
10x10	132	Simple	0.5024	38.7080	1.0047	49.7500
		Kosko	0.5230	26.7380	1.0461	50.0000
		Gaussian	0.5925	39.8500	1.1850	80.6000

Table B.8: Performance comparison of the simple rectangular, Kosko and Gaussian on the QSAR bioddegradation dataset

SOM	Eff.rules	Method	Performance			
			MAE	**MAPE**,%	**RAE**	**Acc** ,%
3x3	18	Simple	0.4380	29.4652	0.9794	47.5829
		Kosko	0.4001	29.8527	0.8947	67.2038
		Gaussian	0.1233	11.0065	0.2758	**86.1611**
5x5	41	Simple	0.3658	22.4755	0.8182	49.2891
		Kosko	0.3695	29.2198	0.8263	68.8152
		Gaussian	0.1281	10.3817	0.2865	85.4028
10x10	71	Simple	0.3582	24.0573	0.8011	49.5735
		Kosko	0.3480	28.1561	0.7784	72.1327
		Gaussian	0.1880	14.0119	0.4204	76.4929

Appendix C

Multinomial Malware Classification - A Novel Dataset

The motivation was to perform a study of different malware categories and families that are available for Windows OS. If we look on previous studies that involve PE32 file formats for MS Windows, we can see that majority only target differentiation between "malicious" and "benign" samples. However, there are quite many malware categories that have different characteristics and functionality. Therefore, our idea is to study how static analysis of PE32 files with help of Machine Learning can facilitate large-scale malware detection into families and categories.

C.1 Acquisition of Raw Characteristics

Malware samples acquisition. There was a malware collection initiative that took place within the Testimon Research group[1] at HiG during April - June 2015. It resulted in a number of samples from students, 10 first archives of Virustotal and files from VxHeaven were collected in addition. After thorough analysis and filtering, we derived all possible PE32 files and removed other types of files. In overall we ended up with 407,741 malware samples, yet some of them will be eliminated. The total size of all the executables is 136GB.

Characteristics acquisition. Since we targeted a static analysis due to large number of samples, it was decided to extract as much characteristic raw data as possible. Two main sources that we used were PEFRAME and VIRUSTOTAL. PEFRAME presents comprehensive set of attributes that can be found in the PE header. There were some works before showing that it can be possible to identify

[1] https://testimon.ccis.no/

malware using such headers information. VIRUSTOTAL presents scan results from over 50 anti-virus databases, information about possible packers and compressors in addition to basis PE headers data. Moreover, we used standard linux tools to get more file characteristics, e.g. size of different sections, strings and also entropy. Finally, we created a MySQL database that contains raw characteristics of the PE32 windows executables (all malware) filtered out of the mess of different malwares that were present in gathered earlier sets. Fields in the SQl dataset are following:

1. *md5* - md5 of the file
2. *virustotal_file_report* - retrieved Virustotal report using private API, serialized JSON-formatted[2]
3. *virustotal_file_behaviour* - retrieved Virustotal report using private API, serialized JSON-formatted[3]
4. *virustotal_file_network_traffic* - retrieved Virustotal report using private API, serialized JSON-formatted [4]
5. *peframe* - Report from executing the peframe script agains the malware, JSON-formatted[5]
6. *file* - output of the Linux command 'file', particularly interesting different architectures and so on.
7. *strings* - output of the Linux command 'strings', we can think of counting the number of strings, etc.
8. *size* - output of the second string of the Linux command 'size', contains information about size of different sections of the binary file

   ```
   text data bss dec hex filename
   105182 2044 3424 110650 1b03a /bin/ls
   ```

9. *file_entropy* - entropy of the file, may indicate strong encryption in case if close to 8.0
10. *size_of_file* - size of the file in bytes.

[2]https://www.virustotal.com/en/documentation/private-api/#get-report
[3]https://www.virustotal.com/en/documentation/private-api/#get-behaviour
[4]https://www.virustotal.com/en/documentation/private-api/#get-network-traffic
[5]https://github.com/guelfoweb/peframe

11. *do_not_process* - just a binary flag, used to indicate which malwares could not be processed by peframe script due to time-out and have to be excluded from further experiments.

The target of the project is to process raw characteristics into numerical features and extract more or less consistent/conventional names of malware families and malware categories. Both can be used later in classification tasks. Malware families defined in work by Microsoft [6]

C.2 List of PE32 Architectures

The top PE32 architectures that can be found in the raw dataset are listed in the Table C.1.

[6] http://www.microsoft.com/security/pc-security/malware-families.aspx

Table C.1: PE32 architectures list from the dataset

Samples	Architecture
231445	PE32 executable (GUI) Intel 80386, for MS Windows
57012	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
50608	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed
11009	PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft Installer self-extracting archive
9349	PE32 executable (GUI) Intel 80386, for MS Windows, PECompact2 compressed
8668	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows, UPX compressed
8507	PE32 executable (native) Intel 80386, for MS Windows
8117	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows
5004	PE32 executable (console) Intel 80386, for MS Windows
4758	PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
1604	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows, PECompact2 compressed
1314	PE32 executable (DLL) (console) Intel 80386, for MS Windows
1134	PE32 executable (GUI) Intel 80386, for MS Windows, Petite compressed
1061	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows, UPX compressed
825	PE32 executable (console) Intel 80386 (stripped to external PDB), for MS Windows
748	PE32+ executable (GUI) x86-64, for MS Windows
743	PE32 executable (DLL) (native) Intel 80386, for MS Windows
733	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows, Nullsoft Installer self-extracting archive
625	PE32 executable (DLL) Intel 80386, for MS Windows
547	PE32 executable (DLL) (GUI) Intel 80386 (stripped to external PDB), for MS Windows
520	PE32 executable (console) Intel 80386, for MS Windows, UPX compressed
431	PE32 executable (GUI) Intel 80386, for MS Windows, InnoSetup self-extracting archive
336	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed, RAR self-extracting archive
220	PE32 executable (GUI) Intel 80386, for MS Windows, RAR self-extracting archive
202	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows, PECompact2 compressed
185	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows
178	PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows
169	PE32 executable (DLL) (console) Intel 80386 (stripped to external PDB), for MS Windows
141	PE32 executable (GUI) Intel 80386, for MS Windows, InstallShield self-extracting archive
139	PE32+ executable (native) x86-64, for MS Windows

C.3 Raw Characteristics

A number of raw characteristic have been collected to make the multinomial malware analysis feasible. The examples are given below.

C.3.1 PEframe

Example of PEframe output as of 2016 showed in the Listing C.1.

Listing C.1: Example of PEframe output in JSON

```
[{
    "Short Info": {
        "Xor": true,
        "Compile Time": "1970-01-01 02:08:16",
        "Directories": [
            "Import",
            "Export",
            "Resource",
            "Relocation"
        ],
        "Hash SHA-1": "27333588c6f2b7c076e9a279bb40fa6d6f9
           afec1",
        "DLL": true,
        "File Size": "21670",
        "Detected": [
            "Packer"
        ],
        "Hash MD5": "000000b4dccfbaa5bd981af2c1bbf59a",
        "Import Hash": "87bed5a7cba00c7e1f4015f1bdae2183",
        "Sections": 2,
        "File Name": "000000b4dccfbaa5bd981af2c1bbf59a"
    }
}, {
    "Digital Signature": {
        "Block Size": 0,
        "Virtual Address": 0,
        "Hash MD5": false,
        "Hash SHA-1": false
    }
}, {
    "Packer": [
        "Upack V0.36-V0.37 (DLL) -> Dwing",
        "Upack V0.28-V0.399 -> Dwing   * Sign.By.
           fly * 20080321",
        "Upack_Patch or any Version -> Dwing",
        "Upack v0.28 - 0.39 (relocated image base - Delphi,
```

```
            .NET, DLL or something else :) -> Dwing (h)"
    ]
}, {
    "Anti Debug": []
}, {
    "Anti VM": []
}, {
    "Xor": true,
    "Offset": []
}, {
    "Suspicious API": [
        "GetProcAddress",
        "LoadLibraryA"
    ]
}, {
    "Suspicious Sections": [
        {
            "Section": ".Upack\u0000\u0000",
            "Hash MD5": "d41d8cd98f00b204e9800998ecf8427e",
            "Hash SHA-1": "da39a3ee5e6b4b0d3255bfef95601890
               afd80709"
        },
        {
            "Section": ".rsrc\u0000\u0000\u0000",
            "Hash MD5": "76baef1d5e7c419db489e7df3bedf462",
            "Hash SHA-1": "b9c1077bb056c92bc2147902b51135e0
               5df8fb17"
        }
    ]
}, {
    "Url": [],
    "File Name": [
        [
            "Library",
            [
                "MZKERNEL32.DLL"
            ]
        ]
    ]
}, {
    "Meta Data": [
        "LegalCopyright: (C) Microsoft Corporation. All
           rights resad.",
        "InternalName: msplay32",
        "FileVersion: 5.1.2600.3099",
```

```
        "CompanyName: Microsoft Corporation",
        "LegalTrademarks: Microsoft",
        "Comments: ",
        "ProductName: Microsoft(R) Windows(R) Operating
            System",
        "ProductVersion: 5.1.2600.3099 (xpsp_sp2_gdr.070308
            -0222)",
        "FileDescription: Windows XP MSPLAY API DLL",
        "OriginalFilename: msplay32",
        "Translation: 0x0804 0x03a8"
    ]
}
]
```

C.3.2 VirusTotal

Example of VirusTotal report showed in the Listing C.2.

Listing C.2: Example of PEframe output in JSON

```
stdClass Object
(
    [vhash] => 12402f0f7bz2?z7
    [submission_names] => Array
        (
            [0] => 000000b4dccfbaa5bd981af2c1bbf59a27333588
                c6f2b7c076e9a279bb40fa6d6f9afec121670.dll
            [1] => msplay32
            [2] => 000000b4dccfbaa5bd981af2c1bbf59a.dll
            [3] => /var/newbot/vh/Trojan-GameThief.Win32.
                OnLineGames.ikb
            [4] => Trojan-GameThief.Win32.OnLineGames.ikb
            [5] => 000000B4DCCFBAA5BD981AF2C1BBF59A
            [6] => 000000b4dccfbaa5bd981af2c1bbf59a
        )

    [scan_date] => 2014-04-04 18:38:08
    [first_seen] => 2011-07-04 16:19:24
    [times_submitted] => 15
    [additional_info] => stdClass Object
        (
            [exports] => Array
                (
                    [0] => DllCanUnloadNow
                    [1] => DllGetClassObject
                    [2] => DllRegisterServer
```

```
                    [3] => DllUnregisterServer
                    [4] => JumpOff
                    [5] => JumpOn
                    [6] => ThreadPro
                )

            [exiftool] => stdClass Object
                (
                    [LegalTrademarks] => Microsoft
                    [FileDescription] => Windows XP MSPLAY
                        API DLL
                    [InitializedDataSize] => 10752
                    [ImageVersion] => 0.0
                    [ProductName] => Microsoft(R) Windows(R
                        ) Operating System
                    [FileVersionNumber] => 5.1.2600.3099
                    [LanguageCode] => Chinese (Simplified)
                    [FileFlagsMask] => 0x003f
                    [CharacterSet] => Windows, Chinese (
                        Simplified)
                    [LinkerVersion] => 0.58
                    [OriginalFilename] => msplay32
                    [MIMEType] => application/octet-stream
                    [Subsystem] => Windows GUI
                    [FileVersion] => 5.1.2600.3099
                    [TimeStamp] => 1970:01:01 02:08:16+01:0
                        0
                    [FileType] => Win32 DLL
                    [PEType] => PE32
                    [InternalName] => msplay32
                    [SubsystemVersion] => 4.0
                    [FileAccessDate] => 2014:04:04 19:39:27
                        +01:00
                    [ProductVersion] => 5.1.2600.3099 (
                        xpsp_sp2_gdr.070308-0222)
                    [UninitializedDataSize] => 0
                    [OSVersion] => 4.0
                    [FileCreateDate] => 2014:04:04 19:39:27
                        +01:00
                    [FileOS] => Win32
                    [LegalCopyright] => (C) Microsoft
                        Corporation. All rights resad.
                    [MachineType] => Intel 386 or later,
                        and compatibles
                    [CompanyName] => Microsoft Corporation
```

```
                    [CodeSize] => 4096
                    [FileSubtype] => 0
                    [ProductVersionNumber] => 5.1.2600.3099
                    [EntryPoint] => 0x16ed7
                    [ObjectFileType] => Executable
                        application
                )

            [trid] => DOS Executable Generic (100.0%)
            [pe-imphash] => 87bed5a7cba00c7e1f4015f1bdae218
                3
            [pe-resource-langs] => stdClass Object
                (
                    [NEUTRAL] => 2
                    [CHINESE SIMPLIFIED] => 3
                )

            [clam-av-pua] => ClamAV PUA (Possibly Unwanted
                Application) detection:
While not necessarily malicious, the scanned file presents
    certain
characteristics which depending on the user policies and
    environment may
or may not configure a threat.
For full details see: http://www.clamav.net/support/faq/pua
            [magic] => PE32 executable for MS Windows (DLL)
                (GUI) Intel 80386 32-bit
            [sigcheck] => stdClass Object
                (
                    [publisher] => Microsoft Corporation
                    [product] => Microsoft(R) Windows(R)
                        Operating System
                    [description] => Windows XP MSPLAY API
                        DLL
                    [copyright] => (C) Microsoft
                        Corporation. All rights resad.
                    [original name] => msplay32
                    [file version] => 5.1.2600.3099
                    [internal name] => msplay32
                    [link date] => 2:08 AM 1/1/1970
                )

            [compressed_parents] => Array
                (
                    [0] => b484f4f5ed824a75c32f765c3d5f85c1
```

```
                    9cfd860ea41cce09c8cc977f7fb2bf61
97              )
98
99          [imports] => stdClass Object
100             (
101                 [KERNEL32.DLL] => Array
102                     (
103                         [0] => LoadLibraryA
104                         [1] => GetProcAddress
105                     )
106
107             )
108
109         [f-prot-unpacker] => UPack, PE_Patch.MaskPE
110         [peid] => WinUpack v0.39 final (relocated image
                base) -> By Dwing (c)2005 (h2)
111         [pe-resource-types] => stdClass Object
112             (
113                 [RT_ICON] => 1
114                 [RT_GROUP_ICON] => 1
115                 [RT_VERSION] => 1
116                 [RT_RCDATA] => 2
117             )
118
119         [pe-timestamp] => 4096
120         [pe-resource-list] => stdClass Object
121             (
122                 [e3b0c44298fc1c149afbf4c8996fb92427ae41
                    e4649b934ca495991b7852b855] =>
                    English text
123                 [513bd37a0ce952e5142954c6373b2d58e7f078
                    bd17d2f52125daaefcb53bee30] => data
124             )
125
126         [pe-entry-point] => 93911
127         [sections] => Array
128             (
129                 [0] => Array
130                     (
131                         [0] => .Upack
132                         [1] => 4096
133                         [2] => 69632
134                         [3] => 0
135                         [4] => 0.00
136                         [5] => d41d8cd98f00b204e9800998
```

```
                                    ecf8427e
                            )

                        [1] => Array
                            (
                                [0] => .rsrc
                                [1] => 73728
                                [2] => 53248
                                [3] => 21106
                                [4] => 7.88
                                [5] => 76baef1d5e7c419db489e7df
                                    3bedf462
                            )

                    )

                [pe-machine-type] => 332
                [command-unpacker] => UPack, PE_Patch.MaskPE
            )

        [size] => 21670
        [scan_id] => 3ab5d534d493e65f01ee5dd12d650091cbf5a5a833
            56cbf3f8abf7ad5350ed72-1396636688
        [total] => 51
        [harmless_votes] => 0
        [verbose_msg] => Scan finished, information embedded
        [sha256] => 3ab5d534d493e65f01ee5dd12d650091cbf5a5a8335
            6cbf3f8abf7ad5350ed72
        [type] => Win32 DLL
        [scans] => stdClass Object
            (
                [Bkav] => stdClass Object
                    (
                        [detected] => 1
                        [version] => 1.3.0.4959
                        [result] => W32.WowStealBDll.Trojan
                        [update] => 20140404
                    )
.....................................................
                [Qihoo-360] => stdClass Object
                    (
                        [detected] => 1
                        [version] => 1.0.0.1015
                        [result] => Trojan.PSW.Win32.WOW.D
                        [update] => 20140404
```

```
                )

        )

    [tags] => Array
        (
            [0] => upack
            [1] => pedll
        )

    [unique_sources] => 9
    [positives] => 46
    [ssdeep] => 384:bWWTEcWWcL9bXUU4Y1tzcz7O/vTHIreN7zNyM3tsbFxJ26IbMpBdGUEf3GR+MAd/:UVhtoHO/vVwZIbvf3BFwvzA
    [md5] => 000000b4dccfbaa5bd981af2c1bbf59a
    [permalink] => https://www.virustotal.com/file/3ab5d534d493e65f01ee5dd12d650091cbf5a5a83356cbf3f8abf7ad5350ed72/analysis/1396636688/
    [sha1] => 27333588c6f2b7c076e9a279bb40fa6d6f9afec1
    [resource] => 000000B4DCCFBAA5BD981AF2C1BBF59A
    [response_code] => 1
    [community_reputation] => 0
    [malicious_votes] => 0
    [ITW_urls] => Array
        (
        )

    [last_seen] => 2014-04-04 18:38:08
)
```

Appendix D

Author's Biography

D.1 Curriculum Vitae

Andrii Shalaginov graduated in 2011 from the National Technical University of Ukraine "Kiev Polytechnic Institute", Faculty "Institute of Applied Systems Analysis", Department of Computer Aided Design, where he pursued his B.Sc. and 1^{st} M.Sc. in System Design. In addition to this he had R& D and industrial experience following the study.

Later on Andrii finished his 2^{nd} M.Sc. in Information Security, Digital Forensics track at the Gjøvik University College in 2013. As a PhD Candidate he joined Digital Forensics Group at the Norwegian University of Science and Technology and is currently doing his research in the area of Computational Forensics under supervision of Prof. Dr. Katrin Franke and Prof. Dr. Slobodan Petrovic. This work is related to a Digital Forensics and Machine Learning that includes malware analysis, intrusion detection and analysis of large-scale digital forensics data. In addition to this Andrii was a work package editor for the SuPLight project responsible for deploying secure collaborative platform for plugins interaction.

In 2015 Andrii was elected as a representative in steering committee for the COINS Research School of Computer and Information Security. During 2013-2017 Andrii has also been participating in a variety of conferences, seminars and summer and winter schools devoted to Information Security and Artificial Intelligence where he presented his research. He also was awarded with a travel grant by Artificial Intelligence journal in 2017. Moreover, Andrii has been actively writing reviews for a number of channels such that book, Cyber Threat Intelligence, ACM Computing Surveys, journal Applied Soft Computing, journal IEEE Transactions on Big Data,

IEEE Privacy, Security and Trust. He also supervised several M.Sc. students with first class honours theses, while the results of research was published in proceedings of international conferences. One of the M.Sc. students, Lars Strande Grini, had received ISACA Norway Chapter's stipent for best MSc thesis in 2016.

D.2 List of Publications

Papers in Conference Proceedings

1. **Shalaginov, Andrii**; Franke, Katrin. A Deep Neuro-Fuzzy method for multi-label Windows PE32 malware classification. *IEEE Symposium Series on Computational Intelligence (IEEE SSCI)*, 2017.

2. **Shalaginov, Andrii**. Fuzzy logic model for Digital Forensics: A trade-off between accuracy, complexity and interpretability. *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

3. **Shalaginov, Andrii**. Evolutionary Optimization of On-line Multilayer Perceptron for Similarity-Based Access Control. *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2017.

4. **Shalaginov, Andrii**. Dynamic feature-based expansion of fuzzy sets in Neuro-Fuzzy for proactive malware detection. *IEEE 20th International Conference on Information Fusion (Fusion)*, 2017.

5. **Shalaginov, Andrii**; Franke, Katrin. Automated intelligent multinomial classification of malware species using dynamic behavioural analysis. *14th Annual Conference on Privacy, Security and Trust (PST)*, 2016.

6. Andersen, Lars Christian; Franke, Katrin; **Shalaginov, Andrii**. Data-driven Approach to Information Sharing using Data Fusion and Machine Learning for Intrusion Detection. *Norsk Informasjonssikkerhetskonferanse (NISK) 2016*; Volume 2016. s. 19-30.

7. Banin, Sergii; **Shalaginov, Andrii**; Franke, Katrin. Memory access patterns for malware detection. *Norsk Informasjonssikkerhetskonferanse (NISK) 2016*; Volume 2016. s. 96-107.

8. **Shalaginov, Andrii**. Soft Computing and Hybrid Intelligence for Decision Support in Forensics Science. *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data*. IEEE 2016 ISBN 978-1-5090-3865-7. s. 304-306.

9. **Shalaginov, Andrii**; Grini, Lars Strande; Franke, Katrin. Understanding Neuro-Fuzzy on a Class of Multinomial Malware Detection Problems. *IEEE International Joint Conference on Neural Networks (IJCNN)*. Research Publishing Services 2016 ISBN 978-1-5090-0619-9. s. 684-691.

10. Grini, Lars Strande; **Shalaginov, Andrii**; Franke, Katrin. Study of Soft Computing methods for large-scale multinomial malware types and families detection, *6th World Conference on Soft Computing*, Berkeley, California, 2016.

11. **Shalaginov, Andrii**; Franke, Katrin. A new method of fuzzy patches construction in Neuro-Fuzzy for malware detection. *Proceedings of the 2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology, Eusflat-15*. Atlantis Press 2015 ISBN 978-94-62520-77-6. s. 170-177.

12. **Shalaginov, Andrii**; Franke, Katrin. Automated generation of fuzzy rules from large-scale network traffic analysis in Digital Forensics Investigations. *2015 Seventh International Conference of Soft Computing and Pattern Recognition (SoCPaR 2015)*. IEEE 2015 ISBN 978-1-4673-9360-7. s. 31-36.

13. **Shalaginov, Andrii**; Franke, Katrin. Automatic rule-mining for malware detection employing Neuro-Fuzzy Approach. *Proceeding of Norwegian Information Security Conference / Norsk informasjonssikkerhetskonferanse - NISK 2013 - Stavanger*, 18th-20th November 2013. Akademika forlag 2013 ISBN 978-82-321-0366-9. s. 100-111.

14. Kiseleva, Aanna; **Shalaginov, Andrii** ; Yamnenko, Yulia. Prediction of the active zone in the control system power consumption using Expectation Maximization procedure. *Proceedings of the II International Conference "Automation control and intelligent system and environment"*, pp. 67-71, 2011, in Russian.

15. **Shalaginov, Andrii**. Cubic spline extrapolation of time series. *Proceedings of the XIII International Scientific and Technical Conference "System Analysis and Information Technologies"*, p. 397, 2011, in Russian.

16. **Shalaginov, Andrii**; Tsios, Sergii; Kadin, Eugeniy. Full text search in knowledge bases of information web portals. *Proceedings of the XIII International Scientific and Technical Conference "System Analysis and Information Technologies"*, p. 514, 2011, in Russian.

17. Kiseleva, Anna; Kiselev, Gennadiy; **Shalaginov, Aandrii**. Application of methods of linear prediction and extrapolation to the context-dependent applications. *Proceedings of the XI Scientific Conference on behalf of T. A. Taran "Intelligent analysis of Information"*, pp. 269-276, 2011, in Russian.

18. Kiseleva, Anna; **Shalaginov, Andrii**. Hidden Markov Model for Dealing with Context Application. *Proceedings of the XVIII Ukrainian-Polish Conference "CAD in Machinery Design. Implementation and Education problems"*, pp. 20-22, 2010.

19. **Shalaginov, Andrii**. Search optimization of institute department web site in the Internet. *Proceedings of the XII International Scientific and Technical Conference "System Analysis and Information Technologies"*, p. 504, in Ukrainian.

20. **Shalaginov, Andrii**. 32-bit applications and 64-bit operating systems. *Proceedings of the Student Conference "Innovations in Technology"*, p. 250.

Chapters in Books

1. **Shalaginov, Andrii**; Banin, Sergii; Dehghantanha, Ali; Franke, Katrin. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. *Cyber Threat Intelligence*, 2017.

2. Wangen, Gaute; **Shalaginov, Andrii**. Quantitative Risk, Statistical Methods and the Four Quadrants for Information Security. *Risks and Security of Internet and Systems: 10th International Conference, CRiSIS 2015*, Mytilene, Lesbos Island, Greece, July 20-22, 2015, Revised Selected Papers. Springer 2016 ISBN 978-3-319-31811-0. s. 127-143.

3. Wangen, Gaute; **Shalaginov, Andrii**; Hallstensen, Christoffer V. Cyber security risk assessment of a DDoS attack. *Lecture Notes in Computer Science 2016*; Volume 9866. s. 183-202.

4. **Shalaginov, Andrii**; Franke, Katrin. A New Method for an Optimal SOM Size Determination in Neuro-Fuzzy for the Digital Forensics Applications. *Advances in Computational Intelligence; 13th International Work-Conference on Artificial Neural Networks, IWANN 2015*, Palma de Mallorca, Spain, June 10-12, 2015. Proceedings, Part II. Springer 2015 ISBN 978-3-319-19222-2. s. 549-563.

5. **Shalaginov, Andrii**; Franke, Katrin. Towards Improvement of Multinomial Classification Accuracy of Neuro-Fuzzy for Digital Forensics Applications.

Hybrid Intelligent Systems - proceedings of 15th International Conference HIS 2015 on Hybrid Intelligent Systems. Springer Publishing Company 2015 ISBN 978-3-319-27220-7. s. 199-210.

Journal Articles

1. **Shalaginov, Andrii**; Franke, Katrin. Big data analytics by automated generation of fuzzy rules for Network Forensics Readiness. *Journal Applied Soft Computing 2017*; Volume 52. s. 359-375.

2. **Shalaginov, Andrii**; Franke, Katrin. Intelligent generation of fuzzy rules for network firewalls based on the analysis of large-scale network traffic dumps. *International Journal of Hybrid Intelligent Systems 2016*; Volume 13.(3-4) s. 195-206.

3. **Shalaginov, Andrii**; Franke, Katrin. Multinomial classification of web attacks using improved fuzzy rules learning by Neuro-Fuzzy. *International Journal of Hybrid Intelligent Systems 2016*; Volume 13.(1) s. 15-26.

4. **Shalaginov, Andrii**; Franke, Katrin; Huang, Xiongwei. Malware Beaconing Detection by Mining Large-scale DNS Logs for Targeted Attack Identification. *World Academy of Science, Engineering and Technology: An International Journal of Science, Engineering and Technology 2016*; Volume 10.(4) s. 617-629.

5. Kiseleva, Anna; Kiselev, Gennadiy; Sergeev, Aalexey; **Shalaginov, Andrii**. Processing the input data in multimodal applications. *Scientific and Technical Journal "Electronics and Communications"*, volume 2, pp. 86-92, 2011, in Russian.

Posters & Selected Talks

1. **Shalaginov, Andrii**. Machine Learning Aided Malware Analysis - Research at NTNU. *NorCERT sikkerhetsforum*; 2017-03-30.

2. **Shalaginov, Andrii**. Application of Computational Intelligence for Digital Forensics. *COINS PhD Seminar*; 2015-10-18.

3. **Shalaginov, Andrii**. Automated generation of the human-understandable rules from network traffic dumps. *3rd National Workshop on Data Science (SweDS)*; 2015-09-30.

4. **Shalaginov, Andrii**; Franke, Katrin. Generation of the human-understandable fuzzy rules from large-scale datasets for Digital Forensics applications using Neuro-Fuzzy. *NordSec 2015*; 2015-10-19.

Technical Reports

1. **Shalaginov, Andrii**; Franke, Katrin. Statistical Analysis of Material Properties. *SuPLight Project*, 2013.

List of Abbreviations

AC	Access Control
AI	Artificial Intelligence
ASCII	American Standard Code for Information Interchange
ANFIS	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Networks
API	Application Programming Interface
APK	Android Package File
APT	Advanced Persistent Threats
AV	Anti-Virus
BMU	Best Matching Unit
BYOD	Bring Your Own Device
C4.5	Name of Decision Tree algorithm
CARO	Computer Antivirus Research Organization
CF	Computational Forensics
CFS	Correlation-based Feature Selection
CI	Computational Intelligence

CNN Convolution Neural Networks

COFF Common Object File Format

CPU Central Processing Unit

DAC Discretionary Access Control

DC Distance Correlation

DDoS Distributed Denial-of-Service Attack

DoS Denial-of-Service Attack

DENF Dynamically-Expanded Neuro-Fuzzy

DLL Dynamic-Link Library

DF Digital Forensics

DNN Deep Neural Network

DT Decision Tree

EBP Error Back Propagation

EC Evolutionary Computing

ECOC Error Correcting Output Coding

EM Expectation Maximization

FIS Fuzzy Inference Systems

FL Fuzzy Logic

FP False Positive

FPR False Positive Rate

FS Feature Selection

HTTP Hypertext Transfer Protocol

GA Genetic Algorithm

GD Gradient Descent

GPU Graphics Processing Unit

GSS Golden Section Search

GUI Graphic User Interface

HDD Hard Disk Drive

HI Hybrid Intelligence

HIGGS Higgs bosons dataset

IAT Import Address Table

ICT Information and Communication Technology

IDS Intrusion Detection System

ISRA Information Security Risk Assessment

JSON JavaScript Object Notation

k-NN k-Nearest Neighbours

KDD Knowledge Discovery in Databases

MAC Mandatory Access Control

MAE Mean Absolute Error

MAPE Mean Absolute Percentage Error

MF Membership Function

MIC Maximal Information Coefficient

ML Machine Learning

MLP Multilayer Perceptron

NB Naive Bayes

NF Neuro-Fuzzy

NN Neural Networks

OS Operating System

OSI Open Systems Interconnection

PC Personal Computers

PCA Principle Component Analysis

PCC Pearson Correlation Coefficient

PE32 Portable Executable 32-bit

PM Probabilistic Modelling

PR Packet Rate

PSO Particle Swam Intelligence

QQ Quantile-Quantile plot

RAM Random-Access Memory

RAE Relative Absolute Error

RL Record Linkage dataset

RMSE Root Mean Square Error

RRSE Root Relative Squared Error

SBAC Similarity-Based Access Control

SC Soft Computing

SI Swarm Intelligence

SOM Self-Organizing Map

SUSY Supersymmetric particles dataset

SVM Support Vector Machine

TR True Positive

TPR True Positive Rate

UCI University of California, Irvine

VM Virtual Machine

WAF Web Application Firewall

XML Extensible Markup Language

List of Glossaries

Features Extraction	A process of deriving features from the raw measurable data or characteristics within a mobile device
Fuzzy Logic	A variant of the classical logic, which uses truth degree for each linguistic variable rather than simple binary true or false statements
Fuzzy Rule	A conditional IF-THEN statements that are composed from linguistic variables
Linguistic Terms	The discrete linguistic variable in fuzzy theory that can have truth degree (instead of classical true or false)
Linguistic Rules	In this study means fuzzy rules used for malware detection
Linguistic Variables	The variables in fuzzy logic theory, which can take linguistic terms as values. In this work security metrics are considered as linguistic variables
Membership Function	In fuzzy logic represents degree of truth that a given value belongs to some fuzzy term
Neuro-Fuzzy	Fuzzy logic theory that uses artificial neural network to derive the estimate and derive the rules
Rules Construction	A process of composing rules from the security metrics consist of two stages: all possible rules extraction and selection of the most relevant rules