A study of Adversarial Machine Learning for Cybersecurity

Afzal-Houshmand, Sam

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

PhD Thesis

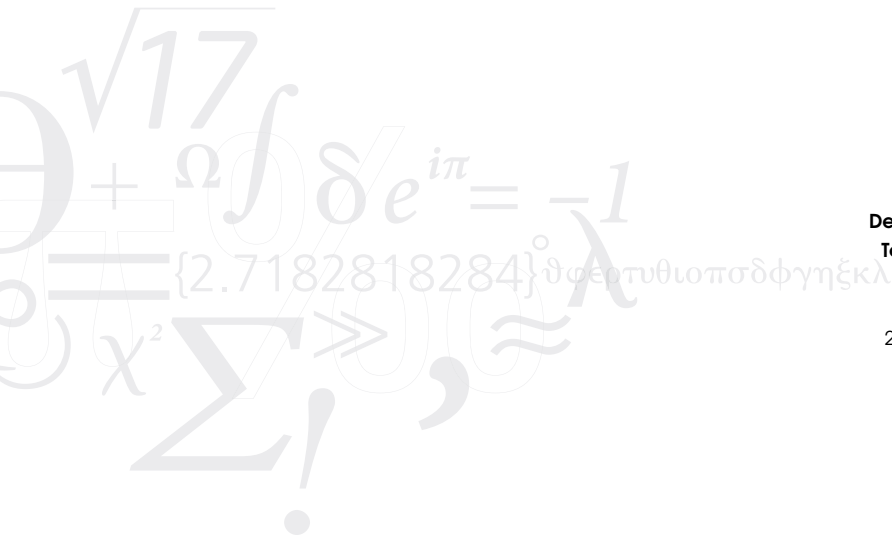
DTU Compute
Department of Applied Mathematics and
Computer Science

A study of Adversarial Machine Learning for Cybersecurity

Sam Afzal-Houshmand

DTU

Supervisor: Prof. Christian D. Jensen
Co-supervisors: Prof. Thanassis Gianetsos



DTU Compute
Department of Compute Science
Technical University of Denmark
Richard Petersens Plads
Building 322
2800 Kongens Lyngby, Denmark

Preface

This thesis was written by Sam Afzal-Houshmand. The subject matter is the PhD conducted on adversarial machine learning for Cybersecurity. It was submitted as part of the requirements needed to obtain a PhD degree at the Technical University of Denmark (DTU). The research was conducted in the section "Cyber Security Engineering" (CSE), Department of Applied Mathematics and Computer science. It was conducted in the time period 1st of October 2019 to 31st of December 2022. This projects was a part of the SecDNS project funded by the Innovation Fund Denmark (IFD) under the SecDNS project (grant number 8090-00050B). The work was done under supervision of Associate Professor Christian D. Jensen and Associate Professor Thanassis Giannetsos, Department of Applied Mathematics and Computer Science, Technical University of Denmark.

Kongens Lyngby, 24th February 2023

Sam Afzal-Houshmand

Abstract

The evolution of Internet of Things is expected to have a major impact on the lives of citizens as new services can be developed by the integration of the physical and digital worlds. However, with this increased richness there is also an emergence of malicious actors. Basically there are hackers that attack the data itself or the models. The models currently in use are heuristics which are susceptible to machine learning techniques. The theory is that to solve the issue of machine learning we use machine learning to defend ourselves i.e. fighting fire with fire. Considering that one of the core advantages is the unprecedented amount of data available for safety-critical decision making, we must be able to control those risks i.e. enabling security towards this direction is the investigation of advanced intrusion detection with the use of machine/deep-learning algorithms capable of achieving enhanced security awareness.

The goal of this research project was to engage artificial intelligence and data science technologies towards developing a unified adversarial classification framework for identifying complex cybersecurity threats on the Internet and other cloud-based networking paradigms, all while taking into account uncertainty of data provenance, used for the classification and simultaneously handling the necessary belief inference and propagation modelling.

Towards this direction, the use of machine learning (ML) and Deep learning (DL) has become omnipresent especially considering the advancements in computational efficiency and the maturity of large datasets available everywhere. Their predictions are used to make decisions about a number of critical applications including (amongst others) security for identifying complex cybersecurity threats in the context of the Internet (e.g., malicious domains) and other networking environments. Concretely, this project will be taking a discourse from the standpoint of malicious Domain Name Server (DNS) detection. DNS is a critical Internet service resolving IP addresses into hostnames that is crucial for the day-to-day operations of most safety critical systems. DNS, however, is susceptible to a wide range of attacks ranging from Domain Hijacking, DNS Flooding and Distributed Reflection Denial of Service (DRDoS) to Cache Poisoning, DNS Hijacking, DNS Spoofing, DNS Tunneling, etc. Leveraging machine- and deep-learning related concepts in the presence of these types of adversarial tactics towards enhanced classification, detection and security awareness is the core pillar of Adversarial Machine Learning - an emerging research area was extensively investigated in the context of this project towards understanding and improving the effectiveness of AI methods in the presence of advanced adversaries.

Streamlined, this project used available collected data sets containing information about known malicious attacks (e.g. phishing, botnets, viruses, man-in-the-middle attack, spoofing etc.), in the context of malicious DNS servers (but not exclusively), towards developing generic classification models that will be able to detect and protect against such advanced adversaries.

The outcomes were frameworks that addressed the central objective of enhancing security against advanced adversaries using ML and DL techniques which were capable of synthetically generate ontologies emulating DNS traffic based on a set of parameters that was determined from research. From this framework a couple of papers emerged which addressed how generating ontologies could be viable and the good results methods such as ML and DL learners applied to such data structure would denote. Furthermore, the viability of using ML and DL techniques in the realm of cybersecurity was investigated using other sources such as Mobile Crowd Sourcing data. This resulted in multiple papers that also take into account advanced techniques in the general area of Artificial Intelligence such as Explainable A.I. With this new understanding and the proofs provided from extensive experimentation there should now be a general approach and viability in applying and investigating ML based models in cybersecurity and comprehending their robustness against advanced adversarial learners.

Resume (Danish)

Udviklingen af Internet of Things forventes at få stor indflydelse på den generelle borgers liv, da nye tjenester kan udvikles ved integration af den fysiske og digitale verden. Men med denne øgede rigdom er der også en fremkomst af ondsindede aktører. Grundlæggende er der hackere, der angriber selve dataene eller modellerne. De modeller, der i øjeblikket er i brug, er heurestik, som er modtagelige for maskinlæringsteknikker. Teorien er, at for at løse problemet med maskinlæring bruger vi maskinlæring til at forsvare os selv, dvs. bekæmpe ild med ild. I betragtning af, at en af kernefordelene er den hidtil usete mængde data, der er tilgængelig til sikkerhedskritisk beslutningstagning, må vi være i stand til at kontrollere disse risici med nøglesikkerhedsaktivere, der er rettet mod denne retning, som er undersøgelsen af avanceret indtrængningsdetektion baseret på brugen af maskine/deep-learning algoritmer der er i stand til at opnå øget sikkerhedsbevidsthed.

Målet med dette forskningsprojekt var at engagere kunstig intelligens og datavidenskabelige teknologier til at udvikle en samlet kontradiktorisk klassifikationsramme til at identificere komplekse cybersikkerhedstrusler på internettet og andre cloudbaserede netværksparadigmer, sammen under hensyntagen til usikkerhed om data herkomst, brugt til klassificeringen, samtidig med at den nødvendige trosslutning og udbredelsesmodellering håndteres.

I denne retning er brugen af maskin- og dyb-læring blevet allestedsnærværende, især i betragtning af fremskridtene inden for beregningseffektivitet og modenheden af store datasæt, der er tilgængelige i moderne sammenhæng. Deres forudsigelser bruges til at træffe beslutninger om en række kritiske applikationer, herunder (blandt andre) sikkerhed til at identificere komplekse cybersikkerhedstrusler i forbindelse med internettet (f.eks. ondsindede domæner) og andre netværksmiljøer. Konkret tog dette projekt en diskurs ud fra synspunktet om ondsindet domænenavneserver (DNS) detektion. DNS er en kritisk internettjeneste, der omsætter IP-adresser til værtsnavne, som er afgørende for den daglige drift af de fleste sikkerhedskritiske systemer. DNS er imidlertid modtagelig for en lang række angreb lige fra domænekapring, DNS-oversvømmelse og DRDoS (Distributed Reflection Denial of Service) til cacheforgiftning, DNS-kapring, DNS-spoofing, DNS-tunneling osv. Udnyttelse af maskin- og dyblæring relaterede koncepter i nærvær af de førnævnte modstridende taktikker mod forbedret klassificering, detektion og sikkerhedsbevidsthed er kernepillen i Adversarial Machine Learning – et andet fremvoksende forskningsområde, der var undersøgt i forbindelse med dette projekt for at forstå og forbedre effektiviteten af AI metoder i

nærværelse af sofistikerede modstandere.

Strømlinet vil dette projekt bruge tilgængelige indsamlede datasæt indeholdende information om kendte ondsindede angreb (f.eks. phishing, botnets, vira, man-in-the-middle-angreb, spoofing osv.) i sammenhæng med ondsindede DNS-servere (men ikke udelukkende) , mod at udvikle generiske klassifikationsmodeller, der vil være i stand til at opdage og beskytte mod sådanne modstandere.

Resultaterne var en ramme, der adresserede det centrale mål, som var i stand til syntetisk at generere ontologier, der emulerede DNS-trafik baseret på et sæt parametre, der blev bestemt ud fra forskning. Ud fra denne ramme dukkede et par artikler op, som omhandlede, hvordan generering af ontologier kunne være levedygtige, og de gode resultater, som sådanne ML- og DL-metoder anvendt på en sådan datastruktur ville betegne. Ydermere blev levedygtigheden af at bruge ML-teknikker inden for cybersikkerhed undersøgt ved hjælp af andre kilder såsom Mobile Crowd Sourcing-data. Dette resulterede i flere artikler, der også tager højde for avancerede teknikker inden for det generelle område for kunstig intelligens såsom Explainable A.I. Med denne nye forståelse og beviserne fra omfattende eksperimenter burde der nu være en generel tilgang og levedygtighed i at anvende og undersøge ML-baserede modeller inden for cybersikkerhed og forstå deres robusthed over for modstridende aktører.

Acknowledgements

I would like to acknowledge my main supervisors Christian D. Jensen and Thanassis Giannetsos whose guidance was very insightful and useful during this period of time where within this PhD was conducted. I would also like to particularity acknowledge Post.Doc Sajad Homayoun for his close collaboration, mentorship and guiding in helping me progress with the PhD throughout.

I would also like to acknowledge our partners and contemporaries at Aalborg Univeristy for their collaboration throughout via the SecDNS project. Their contributions were most helpful in achieving the objective we set out to achieve together for this project.

Another acknowledgement goes out to CSIS security group, in particular Amine Laghaout whose insight and vision for the SecDNS project helped shape the directions we took our research in throughout the PhD.

Additionally, I would also like to thank the Danish Fund of Innovation (IFD) for funding this project and the Technical University of Denmark (DTU) for providing a good space for me to conduct my research within.

Lastly, I would like to thank my friends and family whose support throughout I could not be without.

List of publications

The main publications included in this thesis

- A proof of concept for supervised learning on ontologies
- A Perfect Match: Deep Learning Towards Enhanced Data Trustworthiness in Crowd-Sensing Systems
- Detecting Ambiguous Phishing Certificates using Machine Learning
- Explainable Artificial Intelligence to Enhance Data Trustworthiness in Crowd-Sensing Systems

Abbreviations

ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
IDS	Intrusion Detection System
NIDS	Network Intrusion Detection System
ERD	Entity Relationship Diagram
AML	Adversarial Machine Learning

Contents

CHAPTER 1

Introduction

This PhD was funded and made as part of the larger SecDNS project. The foundation of the SecDNS project was build on designing a comprehensive model for cyber-crime detection based on the use of machine learning concepts and classification algorithms. The central component of the model is the classification of malicious domains, which entail the calculation, aggregation and propagation of beliefs. To that end we would use DNS traffic data that was provided by our collaborators at CSIS security group among others. Particularly, it is beliefs on whether a domain is malicious or not from the basis of the input data features and the type of attack vectors considered during the classification process which were collected with interest.

The part DTU took part in had overall objectives to specify, develop and deliver a belief inference and propagation engine based on a detailed feature constraint analysis and trust management of all data sources (both internal and external). Additional techniques and mechanisms were to be implemented towards scalability in order to extract complex security insights and timely identify evolving threats. Taking all these issues into account, we identified a number of research challenges and defined our methodology on how to tackle each one of them based on the state-of-the-art of each one of these core topics i.e.:

- Statistical Classification Methods and Tools
- Neural-based Classification
- Belief Inference and Propagation
- Adversarial Machine Learning

Specifically, the work conducted in this PhD study was centered around the Adversarial Machine Learning aspects of this project. The desire was to use the foundation of this project to study the viability, effectiveness, mitigation and opportunities possible with adversarial machine learning for cybersecurity. While the SecDNS foundation is found within DNS traffic data, the work done is this thesis including its papers are not constrained to this only. The main objective of the PhD was to investigate and study adversarial machine learning for cybersecurity in its totality. This meant we would take advantage of whatever may have been available to gain the desired knowledge in this broad scope.

This thesis provides context of the main objectives of this study through the theoretical background. This serves as the foundation behind the papers that were

produced and are the central part of this work as we are presenting a collection of papers as the central contributions. By the end of it, it should be clear that the problem disposition that laid foundation for the overall project was addressed effectively through the study in this PhD which in turn lead to contributions in the general field of adversarial machine learning for Cybersecurity from a holistic perspective.

1.1 Methodology for SecDNS

Pertaining to the SecDNS project, belief propagation algorithms are considered the most efficient way of processing calculated beliefs regarding a domain's maliciousness through a network towards any kind of intelligent detection engine. In this context, the SecDNS project has defined an Entity Relationship Diagram (ERD) which serves as the foundation for the research directions and the generalized ontologies that were considered as seen in figure 1.1.

This diagram provided served as the standpoint to build a holistic detection engine capable of identifying malicious domains. In regards to the designed classification and belief inference & propagation capabilities, these can be viewed in two interlocking phases where:

- **Phase I** - is identification and extraction of the data features that best capture

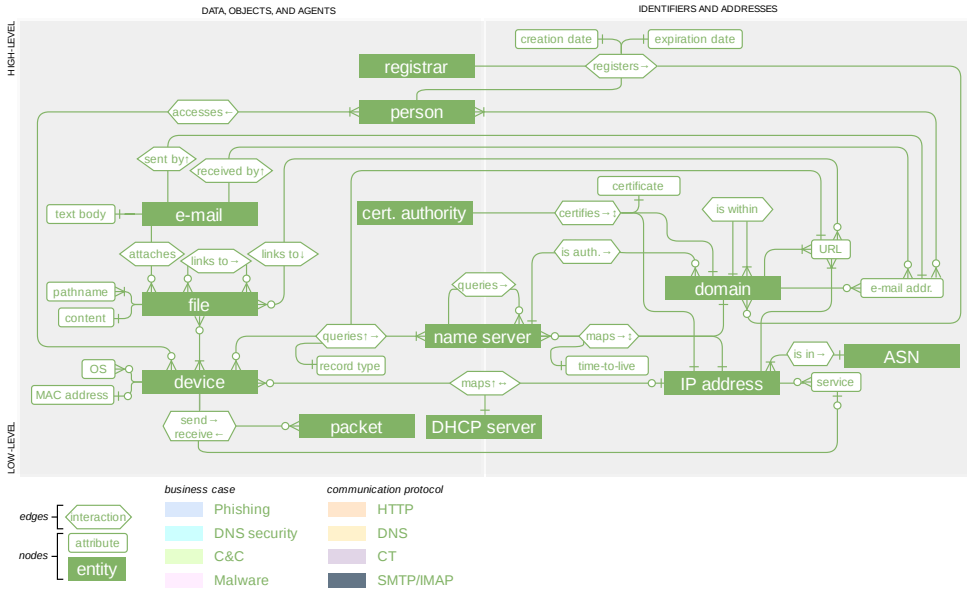


Figure 1.1. ERD from the monograph for the SecDNS project

the modalities and characteristics of the network traffic to be used for the further classification of malicious domains

- **Phase II** - the belief calculation, aggregation, and propagation of the graphical network so that all calculated beliefs can be correlated towards the final classification decision of benign and malicious classification.

The tasks on DTU's end was centered around defining a novel classification model of malicious domains based on a number of evidence (classification based on maliciousness).

In this thesis we will first get into details of what problem disposition we were set with at the onset of this project. Then a theoretical background will be presented to add detailed context to the papers which are the central piece of this work. Then centrally for the thesis, each key paper that this study produced is presented. Lastly, a chapter on the outlook we gained from the outcomes from the papers and project in general is provided.

CHAPTER 2

Problem Dispositions

In this section we will go into details on the problem disposition that was presented at the onset of this project and the other issues that emerged throughout the time period of the study as a result of what was studied about the state-of-the-art in the area. This will lead into the main objectives that were then defined for this study which were attempted to be addressed with the papers and work that this project resulted in.

Traditionally, computer security has relied on conventional stationary measures (from static repositories of intelligence) of defense against cyber-crime actors. Such intelligence is, however, difficult to acquire and does not scale easily to the internet. Even more significant is the difficulty such a stagnant approach have with preventing novel threats. The SecDNS project arose from the realization that these limitations can be overcome by applying machine learning to the cyber security domain. The central idea was the design of a comprehensive model for cyber crime prevention using machine learning, which would be brought to fruition through the collaboration between academic expertise (at AAU and DTU) and the hands-on access and processing of data (at CSIS). The cyber crime prevention engine that was desired was based on three key concepts:

- Feature engineering (i.e., the design of data relevant to cyber security),
- Belief propagation (i.e., the theory which underpins the propagation of threats in an interacting network of entities just as is the case for virtually any event on the internet)
- Machine learning (i.e., the disruptive, heuristic-independent, self-learning computation which unravels crucial correlations between observed events and imminent threats).

In scope of the greater society, the vision of this project was to create a powerful and comprehensive model which could automatically discover a slew of online threats and vulnerabilities, and thereby preventing extreme losses to the Danish industry in particular while legitimizing online businesses worldwide in general.

In terms of academic perspective, the concept was to attain knowledge in the area and investigate its viability and how to create a generalized framework that can be used to applying A.I concepts in the realm of cyber security. Specifically, study

the effectiveness at creating ML based models for cyber crime prevention and how they are affected by adversaries that could conduct attacks using advanced attack strategies (including ML based attack strategies). This also evolved to encompass, how to create an experimentation framework to address difficulties often encountered when doing ML and DL based experiments such as a lack of labelled data.

2.1 State-of-the art in Cyber Crime Prevention

Cybersecurity encompasses a wide range of cyberattacks and cyberdefenses that includes, but is not limited to: intrusion detection, malware detection, preventative security (e.g. access controls, 2-factor-authentication, security training), network monitoring, and associated investigative and remediation efforts. Defending computer networks from unauthorized use and malicious domains has become an increasingly critical challenge for industries, public organisations, and private individuals in most recent years. These vulnerabilities may be hidden undiscovered on computer networks threatening to be the vectors of the next major data breach, service disruption, or illicit appropriation of hardware. Malicious actors exploit these previously unknown, little-known, or unpatched vulnerabilities to misuse network resources or edge devices, sometimes to costly or disastrous effect.

The utilization of advanced network scanning and security inference algorithms provide the cornerstones for the development of next-generation advanced Network Intrusion Detection Systems (NIDS). By using large data sets and ML techniques to train such systems, the analysis of network traffic and the subsequent detection of security threats are made possible. Conventional NIDSs, antivirus software, and similar tools have relied on heuristic-based rules and malware signatures to perform detection so far. Using Matching, or signature-based detection techniques like these are effective when similar attacks are known to security software developers. However, Advanced actors will sometimes rely on novel, or zero-day, vulnerabilities that are typically resistant to signature-based detection. Detecting malicious events on a network that are previously unknown or for which detection rules has not yet been created requires that NIDSs follow an anomaly detection-based approach typically denoted as a model of ”normal network activity” that is estimated, and future activity on the network is evaluated with respect to its probability under the learnt model. This way, it is possible to distinguish a typical and potentially malicious network activity (i.e., malicious domain) from benign.

As commonly used and despite the significant advances in NIDS technology, the vast majority of solutions still operate using less capable signature-based techniques, contrary to anomaly detection techniques. There are several reasons for this reluctance to switch, which includes the high false error rate (and associated costs), difficulty in obtaining reliable training data, longevity of training data and behavioural dynamics of the system. Shone et al. [1] identify three main challenges concerning the creation of a widely accepted anomaly detection technique capable of overcoming

limitations induced by the ongoing changes occurring in modern networks i.e.:

- The continuous drastic growth in the volume of network data. This growth can be predominantly attributed to increasing levels of connectivity, the popularity of the Internet of Things (IoT) and the extensive adoption of cloud-based services. Dealing with the sheer volumes of Big Data requires techniques that can analyse data in an increasingly rapid, efficient and effective manner.
- The in-depth monitoring and granularity required to improve effectiveness and accuracy. NIDS analysis needs to be more detailed and contextually aware, which means shifting away from abstract and high-level observations. E.g. behavioural changes need to be easily attributable to specific elements of a network such as individual users, operating system versions or protocols.
- The number of different protocols and the diversity of data traversing through modern networks. This is possibly the most significant challenge and introduces high-levels of difficulty and complexity when attempting to differentiate between normal and abnormal behaviour. It increases the difficulty in establishing an accurate norm and widens the scope for potential exploitation or zero-day attacks.

There are four basic types of analytics, providing businesses with insights that can support decision making, which are increasing in both complexity and power i.e.:

- Descriptive
- Diagnostic
- Predictive
- Prescriptive

(i)

In recent years, one of the main focuses within NIDS research and the insights extraction from cyber-threats have been the application of deep learning (DL), machine learning (ML) and shallow learning techniques such as PCA, Naive Bayes, Decision Trees and Support Vector Machines (SVM) [1], [2], [3]. The application of these techniques has offered improvements in detection accuracy. However, there are limitations with these techniques, such as the comparatively high level of human expert interaction required, where expert knowledge is needed to process data e.g. identifying useful data and patterns. Not only is this a labour intensive and expensive process but it is also error prone. Similarly, a vast quantity of training data is required for operation (with associated time overheads), which can become challenging in a heterogeneous and dynamic environment.

To address the above limitations, a research area currently receiving substantial interest across multiple domains is that of Deep Learning (DL) and Adversarial Machine Learning [4]. All the technical advancements in security brought by Artificial

Intelligence are due to its self-learning and self-enhancement capabilities, as it capable of mining and learning various types of data, such as spam e-mails, messages, and videos, to then evolving an autonomous detection/defence systems. Continuous self-training can further promote the performance of AI-powered systems, which includes their stability, accuracy, efficiency, and scalability. This is an advanced subset of ML, which can overcome some of the limitations of shallow learning. Thus far, DL research has demonstrated that its superior layer-wise feature learning can better or at least match the performance of shallow learning techniques [5]. It enables to facilitate a deeper analysis of network data and faster identification of any anomalies.

Recent advances on the DL front include experimental works using Recurrent Neural Networks (RNN) to analyse log and network traffic data [5], [6], [7], [8], which is sequential by nature. These works not only demonstrate the capacity for anomaly detection via RNNs with such data but also introduce an attention mechanism to the model that provides context for the flagged anomalies. These works follow an earlier application of Long Short-Term Memory (LSTM) models for intrusion detection given system call logs captured at the host level [9]. Additionally, models based on Artificial Neural Networks (ANNs), such as Autoencoders [10], [11] and Autoencoder Ensembles, have proven to be effective in detecting anomalous network behaviour while also being lightweight, leading to the creation of a new paradigm of online, unsupervised, distributed and efficient DL-enabled NIDSs, such as Kitsune [12].

2.2 Objectives

Based on the condition of the state-of-the-art in the area along with the requirements set by the SecDNS project, what was needed was the development of algorithms for advanced analytics that will ingest security data towards the realisation of an advanced intrusion framework that would be capable of detecting organisational communication abnormalities and as well as communication abnormalities with the Internet at large through observing traffic from both unused IP address spaces (the so-called "Darknet" or "Deepnet") as well as from used IP address spaces (or Internet). Detection of darknet traffic can be used as an indication of Distributed Denial of Service (DDoS) reflective attacks, misconfigured nodes or malware scans and attacks. On the other hand, anomalous traffic observed in the Internet can indicate a strong likelihood of the presence of compromised or malicious devices in an operational network. Aggregated data can be used for training purposes to detect malicious behaviour. The identified malicious behaviour can also be automatically categorised and stored in a fingerprinting database. This will allow us to successfully identify future attacks with the same or similar behaviour. ML and DL have been used for such classification, although new, emerging research on using Broad Learning based on organically growing network consisting of various kinds of IoT sensors and devices, are showing promise. The data mining processes will be streamlined, creating both high-performance lightweight distributed and accurate models, leading to high detec-

tion accuracy, cost and latency reduction. To this end, Deep Analysis functionalities need to be investigate towards dynamically performing intrusion detection, extracting analytics (features) from network traffic, identifying and correlating anomalous communication at networking and endpoint level, and discovering new patterns by building incremental models that can be parallelized.

Summarily, we can in short claim we strived to:

- Create a framework functioning as a ML based intelligence detection engine using DNS traffic data
- Create an general experimental framework that could address persistent issues regarding modelling NIDS which would allow users to set the parameters themselves and generate neccesary data if needed
- Study the effect of adversarial strategies and the advantages that ML and DL based models may be provide through research and experimentation
- Explorer various techniques of ML and DL approaches in various contexts in the realm of Cybersecurity to attain indication of the general viability of their applications

CHAPTER 3

Theoretical Background and Context

The theoretical background and Context chapter of this thesis is based on the theory behind multiple project deliverables and papers that spun out of the SecDNS project as an outcome of the PhD study. It strive to cover the central topics of research as it concerns this PhD study, and while it won't cover everything it will cover what is needed to understand the context in which the outcomes of this PhD was achieved which mainly reflected in the included papers. This encompass research, proposed methodologies and experimentation which all hopefully lead into the specific outcomes refelected in the included papers.

3.1 Adversarial Machine Learning

The rapidly expanding adoption of ML technologies has rendered them attractive targets for adversaries who want to manipulate such mechanisms for malevolent purposes [13]. All ML systems are trained using data sets that are assumed to be representative and trustworthy for the subject matter in question, thus, enabling the construction of a valid system perception of the matter of interest. However, malicious actors can impact the decision-making algorithms of such approaches by either targeting the training data or forcing the model toward their desired output, e.g., misclassification of abnormal events. These types of attacks, known as *poisoning* and *evasion* attacks [14], allow adversaries to significantly decrease overall performance, cause targeted misclassification or bad behaviour, and insert backdoors and neural trojans [15, 16].

Adversarial Machine Learning (AML) sits at the intersection of machine learning and cyber security, and it is often defined as the study of effective machine learning techniques against an adversarial opponent. For example, Huang et al. [17] proposed a new method that learns robust classifiers from supervised data by generating

adversarial examples as an intermediate step facilitating the attack detection. Representative examples of applications that AML can be applied to include intrusion detection, spam filtering, visual recognition and biometrics authentication.

3.1.1 Threat Model

The attack models considered in the context of AML implement mainly a couple of types of attacks i.e.: *poisoning* and *evasion*. The high level goal of these models is to maximize the generalization error of the classification and possibly mislead the decision making system towards desired malicious measurement values T. As stated in [18], a system that uses machine learning aims to find a hypothesis function f that maps observable events into different classes.

To exemplify such models, we could consider a system that monitors network behaviour and performs anomaly-based intrusion detection. An instance of this behaviour is an event that is classified using utility function f either as Normal or Malicious. Within this context, we can assume an input space $\mathcal{X} = \{x_i\}$ and an output space $\mathcal{Y} = \{y_i\}$, where x_i is an event and y_i is the output of this event determined by f, i.e. $f(x_i) = y_i$. Additionally, we assume that the system has been trained using N samples that form the training set $\mathcal{S}$ and it has derived the system perception, denoted by $\hat{y}$. After the end of the training phase, the system receives new events from the actual environment and classifies them. We define this as the *run-time phase* of the system. For every new event $\hat{x}_i$, f gives a new output $f(\hat{x}_i) = \hat{y}_i$. The following cases will be denoted:

- If $\hat{x}_i$ is malicious and the system does not recognize it as such (false negative) there is a loss l caused to the system.
- If $\hat{x}_i$ is malicious and the system recognizes it as such (true positive) or it is not malicious then there is no loss to the system.
- If $\hat{x}_i$ is not malicious and the system recognizes it as such (false positive) then there is a loss λ.

The aim of the attacker is to maximize the impact the attack has to the system by maximizing $|f(\hat{x}_i) - y_i|$. Thus, a challenge of the system that defends is to find a hypothesis/utility function that minimizes the losses, measured as the distance of $f(\hat{x}_i)$ to the real output y_i. This function can be linear or nonlinear and be more complex in formulation as in [19].

Evasion attacks: The adversary can take on an *evasion* attack against classification during the testing phase thus producing a wrong system perception. In this case, the goal of the adversary is to achieve misclassification of some data towards, for example, remaining stealthy or mimicking some desirable behaviour. With regards to *network anomaly-based detection*, an intrusion detection system (IDS) can be evaded by encoding the attack payload in such a way that the destination of the data is

able to decode it but the IDS is not leading to a possible misclassification. Thus, the attacker can compromise the targeted system being spotted out by the IDS.

An additional goal of the attacker could be to cause "concept drift" to the system leading to continuous system re-training, thus, significantly degrading its performance [13]. To exemplify, we can consider a monitoring system that is based on heterogeneous, crowdsourced data originating froms users' sensing-capable devices towards building a wide-scale information collection networks that can provide insights for various contexts [20] (e.g., traffic jams). In this context, an adversarial agent could introduce a misleading measurement sample at time τ, which can be modelled as:

$$x_i^{(a)}(\tau) = \min(x_i^t + \tau\delta + \eta_i(\tau), T), \quad \forall\, i = \{1, ..., N_a\}, \tag{3.1}$$

such that

$$\begin{aligned} \tau = 0 \qquad & x_i^{(a)} = x_i^t + \eta_i(0) \approx x_i^t \\ \tau = 1 \qquad & x_i^{(a)} = \min(x_i^t + \delta + \eta_i(1), T) \\ \tau = 2 \qquad & x_i^{(a)} = \min(x_i^t + 2\delta + \eta_i(2), T) \\ \cdots & \\ \tau \to \infty \qquad & x^{(a)} = T \end{aligned} \tag{3.2}$$

where δ is a small positive number used to slowly drift from the "legitimate" measurement of the adversarial agent's device to the targeted final malicious measurement value T, and $\eta_i(\tau)$ is a small noise component modelled by Gaussian distribution at time instant τ. The system provides N samples $\{x_i\}$ made up of legitimate measurements $\{x_i^t\}$ and malicious measurements $\{x_i^{(a)}\}$ generated by N_a adversarial agents with the overall goal to trigger the concept drift detection module, thus, leading to a continuous system re-training, hearkening it all back to that re-training inherently performs clustering for unsupervised classification problems which is a very computationally expensive process. When this is achieved, the adversary may also have the chance to conduct a poisoning attack causing even higher damage to the system.

Poisoning attacks: The adversary can poison the training dataset. To achieve this, the adversary *derives* and *injects* a point to decrease the classification accuracy [21]. This attack has the ability to completely distort the classification function during its training process thus allowing the attacker to define the classification of the system in any way that is desired. The magnitude of the classification error depends on the data the attacker has chosen to poison the training. To exemplify, the adversary may be able to create a dataset of anomalous network-layer protocol behaviour and train an anomaly-based intrusion detection system with a labelled attack dataset as the ground-truth. As a result, the detector will not be able to recognize cyber attacks against this network-layer protocol threatening the security of the underlying system. This attack could be customized to also have a significant impact to the quality of a signature-based intrusion detection system, which is responsible, for example, for detecting malware infecting a system or an infrastructure.

Another particularly insidious attack in this category is the *backdoor or trojan attack*, where the adversary carefully poisons the model by inserting a backdoor key to ensure it will perform well on standard training data and validation samples, but misbehaves only when a backdoor key is present. Thus an attacker can selectively make a model misbehave by introducing backdoor keys once the model is deployed. To exemplify, we can consider the case of assistive driving in autonomous vehicles where a backdoor could cause the model to misclassify a stop sign as speed limit whenever a specific mark has been placed on the stop sign. In this case, however, the model would perform as expected on stop signs without this mark, making the backdoor difficult to detect since users do not know the backdoor key.

To notate this, we can denote by b the benefit of the attacker from attacking the system, where $b = |y_i - \hat{y_i}|$, and by c as the cost of the attacker, which may be associated with the amount of effort the attacker has to put to perform the attack and the risk to get captured.

If we assume that the attacker injects a malicious point x_c into a legitimate dataset D to form a new compromised dataset D'. The goal of the attacker is to maximize the generalization error W during the validation of m data samples, e.g.,: $\max_{x_c} W = \sum_{j=1}^{m} l(y_j, f(\hat{x}_j, W_j)) + \lambda||W||$.

where f is the hypothesis function (i.e., classifier) that has been trained on a contaminated training set, which includes x_c, λ is a regularization co-efficient and $\hat{y}_j = W\hat{x}_j$.

3.1.2 Taxonomy of Attacks

While the implementation details of attacks against machine learning may vary considerably, their individual steps can be broadly classified into two distinct phases: (i) Preparation, and (ii) Manifestation, as illustrated and detailed in Figure 3.1.

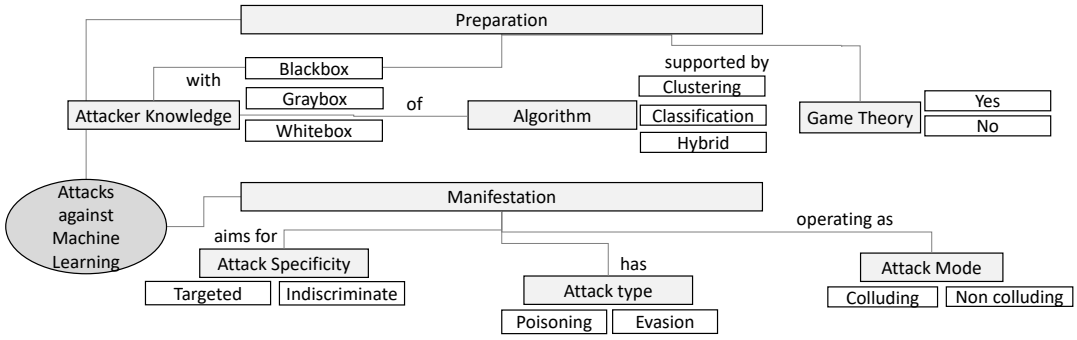


Figure 3.1. A taxonomy of adversarial attacks on machine learning.

3.1.3 Preparation

In the preparation phase, the attackers identify their resources and gather the intelligence required to prepare an attack plan. Here, what determines the characteristics of an adversarial machine learning approach is the knowledge required by the attacker, as well as the type of machine learning technique targeted and whether the attacker is strategic. As such we discuss the following features:

- **Attacker Knowledge**: Here, we take the simplified view whereby the attacker may know (K1) the *Ground truth*, (K2) the learning algorithm, or both, leading to the following attacker knowledge categories:
  - Blackbox attacks: $\neg K_1 \wedge \neg K2$.
  - Graybox attacks: $K1 \vee K2$.
  - Whitebox attacks: $K1 \wedge K2$.

  According to [22], the attacker knowledge may refer to (i) the training data, (ii) the feature set, (iii) the machine learning algorithm along with the objective function minimized during training and (iv) any trained parameters if applicable.

- **Algorithm**: A large variety of machine learning techniques has been targeted in the literature. Commonly, it is DNNs and Convolutional Neural Networks (CNNs) that are typically addressed in the image recognition domain, while in spam email detection, though more common are Naive Bayes, Support Vector Machines (SVM) and Logistic Regression (LR). Other techniques, such as K-Means, K-Nearest Neighbour (KNN), Linear Regression, Community Discovery and Singular Value Decomposition, are typically seen in the malware detection, biometric recognition and network failure and security breach detection domains. For the purposes of this taxonomy, we have classified the techniques based on the machine learning algorithm used in: i) clustering, ii) classification, or iii) hybrid fashion.

- **Game theory**: Adversarial machine learning is commonly equipped with a strategic element, whereby, in game theory terminology, the defender is the machine learning classifier and the attacker is a data generator aiming to contaminate, for example, the training dataset. Both choose their actions strategically in what can be seen as a non-cooperative game [23]. The adversary aims at confusing the classification or clustering with costs related to, e.g. the transformation process or probability of being detected. Contrarily, the defender incurs, for instance a cost for misclassifying samples. The importance of game theory for the defender lies within the field of making the classifiers more aware of adversarial actions and more resistant to them.

3.1.4 Manifestation

Manifestation is the phase where the adversary launches the attack against the machine learning system. Largely dependent on the intelligence gathered in the preparation phase, the attack manifestation can be characterized based on the following characteristics:

- **Attack Specificity**:

  This refers to range of data points that are targeted by the attacker [18, 24]. It is also mentioned as *error specificity* in the recent survey of Barreno et al. [22].

  - **Targeted**: The focus of the attack is on a particular sample (e.g., specific spam email misclassified as legitimate) or a small set of samples.
  - **Indiscriminate**: The adversary attacks a very general class of samples, such as "any false negative" (e.g., maximizing the percentage of spam emails misclassified as legitimate).

- **Attack Type**: This refers to how the machine learning system is affected by an attack [18, 24].

  - **Poisoning**: Poisoning attacks alter the training process through influence over the *training* data.
  - **Evasion**: Evasion attacks exploit misclassifications but *do not affect training* (e.g. the learner or offline analysis, to discover information).

- **Attack Mode**: The original assumption of adversarial machine learning, which is still taken in most related literature, is that attackers work on their own (*non-colluding* case). The alternative is that different *colluding* attackers can collaborate, not only to cover their tracks but also to increase efficiency.

3.1.5 Attack Evaluation

The output of an attack's manifestation is primarily characterized by the nature of its impact on the *accuracy* of a machine learning approach.

- **Evaluation Approach**: A goal of this work is to help researchers and developers improve the resilience of their mechanisms against adversarial machine learning by adopting approaches that have been thoroughly evaluated. We classify the related literature based on whether the proposed approaches have been evaluated *analytically*, in *simulation*, or *experimentally*.

- **Performance impact**: The primary aim of adversarial machine learning is to reduce the performance of a classification or clustering process that is based on machine learning. For classification problems, this can be interpreted as increase in false positives or in false negatives, or in both. For clustering problems, the aim is generally to reduce accuracy.

- **False positives**: In classification problems, such as spam detection, where there are two states (spam or normal), the aim of an attacker may be to make the targeted system falsely label many normal emails as spam emails. This would lead to the user missing emails.
- **False negatives**: Using the same example, if the attacker aims to increase the false negatives, then many spam emails would go through the user's filters.
- **Both false positives and false negatives**: Here, the attacker aims to reduce the overall confidence of the user in their spam filtering system by letting spam emails go through and by filtering out normal emails.
- **Clustering accuracy reduction**: Compared to classification, the accuracy of clustering is less straight-forward to evaluate. Here, we include a general reduction of accuracy as the overall aim of the attacker of a clustering algorithm.

3.2 Embedding Approaches

The datasets we often encounter in the realm of Adversarial machine learning and more specifically DNS traffic data are often not pre-processed. Within that, a major challenge emerged especially in context of DNS Traffic dataset which can be applied to a general issue that can be encountered when creating Intrusion Detection System (IDS) in association to adversarial machine learning models. Namely, when dealing with data which have updating beliefs over time it can be problematic if the entities and interaction between entities are present in nested data structures.

An exammple of nested data structures could be following teh structure as shown in figure 3.2.

An actual excerpt of our DNS dataset in the SecDNS project show the complexity that might be encountered:

Data Structure	Example
$[x]$	[”127.0.0.1”]
$[x_1, x_2, x_3, ..., x_n]$	[”89.202.139.130”, ”89.202.139.136”]
$\{k : v\}$	{”127.0.0.1”: ”no_result”}
$\{k_1 : v_1, k_1 : v_2, k_3 : v_3 ... k_n : v_n\}$	{”2001:4860:4802:32::15”: ”US”, ”2001:4860:4802:34::15”: ”US”}
{k:$[v_1, v_2, v3 ... v_n]$	{”playtv.fr”: [”104.26.14.247”, ”104.26.15.247”, ”172.67.69.167”]}
{k_1:$[v_1, v_2, v3 ... v_n]$, $k2$:$[v_1, v_2, v3 ... v_n]$	{”playtv.fr”: [”104.26.14.247”], ”playtv2.fr”: [”172.67.69.137”] }

The issue that is to be addressed is, that machine learning algorithms typically ingest data in the form of numerical vectors, i.e., of ordered collections of booleans, integers, or floats. In order to engineer such vector features, also known as *embeddings*, one needs a transformation from an arbitrary nesting to a vector.

```
d = list([
        dict(
            key11=list([
                79,
                70,
                41]),
            key12=list([
                96,
                90]),
            ),
        dict(
            key21=list([
                10,
                13,
                17]),
            key22=list([
                26,
                31]),
            key23=list([
                12,
                3]),
            ),
        dict(
            key31=list([
                10,
                10,
                9]),
            ),
        ])
```

Figure 3.2. Example of `list` ∘ `dict` ∘ `list` ∘ `int`.

3.2.1 Embedding literature

Fundamentally, the approaches considered could be broken down into Flatten/collapse the dicts for a single representation that can be embedded where we embed each element and infer the embeddings onto each other via arithmetic operation or infer the embeddings onto each other by feeding them to a Neural network and concatanate the output.

Works that address the specific issue in regards to similar datasets as to DNS traffic are sparse at best. The literature research has touched upon a lot of fairly common technical approaches including embedding of categorical data, one-hot encoding, vector calculations etc. To address the specific issues there seemed to be the most overlap with **word-based encoding techniques** such as word2vec [25], bag-of-words [26] or methods that can also be found via [27]. In this context, however, it was more

advantageous to look more holistic approaches such as doc2vec (or sentence2vec) that used context words to merge embedding of multiple strings or cat2vec (categorical variable to vectors) which allows us to represent high cardinality categorical variables using low dimension embedding. The idea with cat2vec is to turn category into a vector representation by using contextual information to assign a values, which lets you pool information between categories with similarity. In context of NN we can initialize with random vectors and fit the embedding via back-propagation, however, we can also separately generate some embedding and use them effectively with linear or logistic regression [28] [29].

The embedding aspect is fairly approachable but we had investigated approaches on how to effectively combine multiple embedding across multiple stages. These approaches often combine the embedding by using **order, rank or hierachy** or flattening the datasets with no order considered. This came to include multilevel optimizations [30], nested list comprehension [31], dimensionality reduction [32], NMF, multigrid solvers, multiplying vector of embedding, **average of vector embedding, concatenating embedding**, Cosine-similarity. Pairwise rank learning [33], sub list embedding and more. Other approaches include **Graph-based** approaches such as the ones discussed across in stellargraph [34] including various iterations of GCN via vector embeddings [35], nearest neighbor, clustering and classification for reaching the decision [31] which can circumvent some of the issues such as the need to consider rank and averaging the embeddings

The issues to address are two-fold. One issue is to address the high cardinality of the dataset in question and the other issue is how to merge multiple embeddings. To address the high cardinality it seems that a common approach to use is **target encoding** along with factorization [36] which is proven across many works to be suitable for high cardinality categorical string variables such as IP, DNS, URL etc. which is appropriate for us. Other techniques such as Rare-word collapse, frequency binning, random embedding, hashing, luduan features and quantile encoding has proven useful to address high cardinality [37] [38], however, it should be noted that one-hot encoding is typically not recommended for such contexts.

Addressing the second major issue can be more open-ended as some of the more relevant works as denoted earlier take the approach of embeddings being treated as vectors and can effectively be used as representative when computing the dot product or sum of two vectors of embeddings. Another approach that have been suggested across multiple works is to average multiple embeddings and then concatenating them in nested data structures [39] [31]. In machine learning context a normalisation step using e.g. min-max/l1/l2 normalisation can be recommended. Using max instead of average can also be useful. However, in ML if we are using dissimilar features it may be best to use concatenation before embedding [36] i.e. **Average then concatanation**. This is typically done in conjunction with flattening the vectors dimensionality which can be done multiple ways e.g. PCA, SVD, clustering,rank, decision tree, nested list comprehension, GlobalAveragePooling1D, NMF [40].

Another major approach seem to be collapsing the ranks of the dataset to reach a point where a single vector of embedding remain as representative. This can be done

if we can justify similarity.

The average is a good summary because, under a reasonable statistical model of neural embeddings there is a very small chance that two unrelated collections will have similar means. The mean is a reasonable way to summarize embeddings thanks to the concept of dimensionality reduction. Because an exponential number of embeddings are nearly orthogonal in high dimensions, it is unlikely for two independent collections to have similar averages. On the other hand, two related documents are guaranteed to have similar averages. As a result, two collections have similar averages if and only if they share many similar embeddings with very high probability. Retaining order while reducing dimensionalty via common processes such as PCA is also often applied such as themed in figure 3.3.

The intuitive simple way to construct embedding of complex data structures from word embeddings is vector arithmics on all the vectors. Another possibility that can be of interest is to use a fixed operator for vector summarizing like averaging and learn word embeddings in a preceding layer, using a learning target that is aimed at producing rich document embeddings e.g. using sentences to predict context sentences. word embeddings are optimized for averaging into document representations using a cosine layer before softmax pooling of averages of word embeddings [41]. Kanter et al did exactly that, using a simple neural network over an averaging of word vectors, learning the word embeddings by predicting, given a sentence representation, its surrounding sentences. They compare the results to both averaged word2vec vectors and to skip-through vectors.Furthermore, it should be explained that Kanter et.al. proposes an approximate approach to the issues of nested data structures however

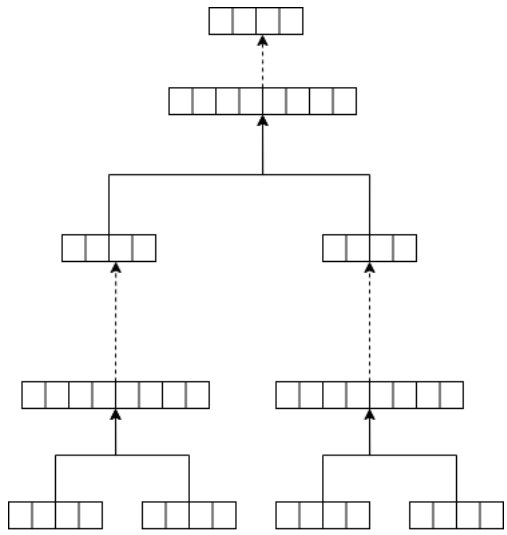


Figure 3.3. Idea is to concatenate to retain order and do dimension reduction operations to gain a vector of fixed length and this diagram illustrate the process in a nested structure

they are still working on a 1-dimensional plane i.e. they concatenate nested values directly unto the the other leafs, which in turn undermines the ordinal info that could be extracted. An approach to inferring the nested values can possible be found in theorems regarding multilevel optimization [31],hierachical graphs [30], pairwise ranking [33] and/or ngraph based appraoches which we will address in their own section. Hill et. al. compare a plethora of methods, including training CBOW and skip-gram word embeddings while optimizing for sentence representation (here using element-wise addition of word vectors) that was based on Mikolov's word2vec model. The skip-gram model computes surrounding word-embeddings by being a model similar to word2vec where is trained on context-target words given by a window-size (words that can appear next to each other). By using one-hot encoding of words as a input we can use the one-hot encoding of a context word and feed it to the neural network. This in turn is multiplied with the weights of the hidden layers in the network. By applying a softmax layer for the output we end up with a probability vector where each component represent the probability of the given string that had been encoded being the defined target word. Afterward a loss function is applied and through back-propagtation we can update the weights which the word-embeddings.

The last point I would like to add is that since there is no logic behind how we order the data in our dataset, there is no advantage to use concatenation as the only reason to use concatenation would be to retain order information.

3.2.2 Graph based embedding

Graph embeddings are the transformation of property graphs to a vector or a set of vectors. Embedding should capture the graph topology, vertex-to-vertex relationship, and other relevant information about graphs, subgraphs, and vertices. We can roughly divide embeddings into two groups [42]:

Vertex embeddings: We encode each vertex (node) with its own vector representation. We would use this embedding when we want to perform visualization or prediction on the vertex level, e.g. visualization of vertices in the 2D plane, or prediction of new connections based on vertex similarities.

Graph embeddings: Here we represent the whole graph with a single vector. Those embeddings are used when we want to make predictions on the graph level and when we want to compare or visualize the whole graphs, e.g. comparison of chemical structures. Later, we will present a few commonly used approaches from the first group (DeepWalk, node2vec, SDNE) and approach graph2vec from the second group.

Graph analytic can lead to better quantitative understanding and control of complex networks, but traditional methods suffer from high computational cost and excessive memory requirements associated with the high-dimensionality and heterogeneous characteristics. Graph embedding techniques can be effective in converting high-dimensional sparse graphs into low-dimensional, dense and continuous vector spaces, preserving maximally the graph structure properties. Another type of emerging graph embedding employs Gaussian distribution-based graph embedding with

important uncertainty estimation. The main goal of graph embedding methods is to pack every node's properties into a vector with a smaller dimension, hence, node similarity in the original complex irregular spaces can be easily quantified in the embedded vector spaces using standard metrics [42].

Node2vec is used for the computation of node embedding, where the embeddings are learned in a way that ensures that the nodes are close in the graph remain close in the embedding space. This is suppported by frameworks such as Stellargraph. [34] Attri2vec is used to calculate node embeddings, where it learns node representations through performing a linear/non-linear mapping on node content attributes while simultaneously making nodes sharing similar context nodes in random walks have similar representations. Once in possession of the node content features we can use those to learn node embeddings where we wish that Attri2vec can achieve better link prediction performnance than the only structure preserving network embedding algorithm Node2vec [43].

GraphSAGE method learns the node embedding for attributed graphs through aggregating neigboring node attributes. The aggregation parameters are learned through encouraging node pairs co-occuring in short random walks to attain similar representations. Node attributes are also leveraged, so graphSAGE is expected to perform better than Node2Vect in link predictions [34].

GCN is yet another method of node embeddings where withing the node embeddings are learned thorugh graph convolution. Conventional GCN relies on node labels as a supervision to perform training, where we consider the unsupervised link prediction setting and attept to kearn informative GCN node embeddings by making nodes co-occuring in short random walks represented closely as is performed in training GraphSAGE [43].

Also worth mentioning is graph embedding approach which embeds the whole where the methods computes one vector to describe the graph namely the graph2vec approach. Graph2vec is based on doc2vec which in turn is based on word2vect. This methods uses a skip-pgra where it gets an id of the data structure in question is trained to predic random occurences. It consists of sampling and relabelling all subgraphs from graph, train the skip-gram model and compute embedding as a one-hot vector retrieving the outcome from the hidden layer [42].

We can use these types of node embeddings to train and evaluate the resulting link prediction model doing a graph mapping of the onthology would produce. This is somewwhat achieved by using 4 different operators i.e. Hadamard, L1, L2, Average. A paper provides a detailed descitption of the these operators. All operators produce link embeddings that have equal dimensionality to the input node embeddings.

One advantage of treating the issue of nested categories as a graph issue is that we can do the merging before the embedding unlike the word-based embeddings which we approach semantically with models such as ELMO. Essentially, we can merge nodes from different graphs into a singular representation hereafter we do the embedding by methods such as Node2Vec via StellarGraph. To merge nodes from different graphs we require a type of compatibility evaluation of sorts. There is tools for that including gvpr which stands for graph pattern scanning and processing language. This will take

graphs as input and output a new graph which has reduced the dimensionality of the two graphs together. This works on the assumption that there is significant similarity between the two graphs.

3.2.3 Nested data structures

The above described approaches are mostly based on data structures with two levels. However, what about cases with multiple levels? Works touch upon this issue namely with **pairwise ranking, hierachical structures, multi-level procedures and Graph based approaches**. It is possible with arithmic approaches we discussed first, specifically, when we apply some type of weight to the inference meaning we can use theorems from advanced aggregations such as hierachical weighted average where multiplication of non-null child data values by their corresponding weight values, then divides the result by the sum of the weight values. Unlike Weighted Average, Hierarchical Weighted Average includes weight values in the denominator sum even when the corresponding child values are null. That why we can infer back the average knowledge of each end leaf. However, for simplicity most works try to apply some sort of dimensionality reduction using SVM, PCA, NMF etc. based upon the contextual information from their data. Hierarchical interpertation of data structures is detailed and illustrated in figure 3.4.

In cases with words in alternate sets e.g. dictionaries it may be advantageous to take an approach that does not rely on distance /similarity between words but rather should be based on devising a mapping-transformation between two words which is

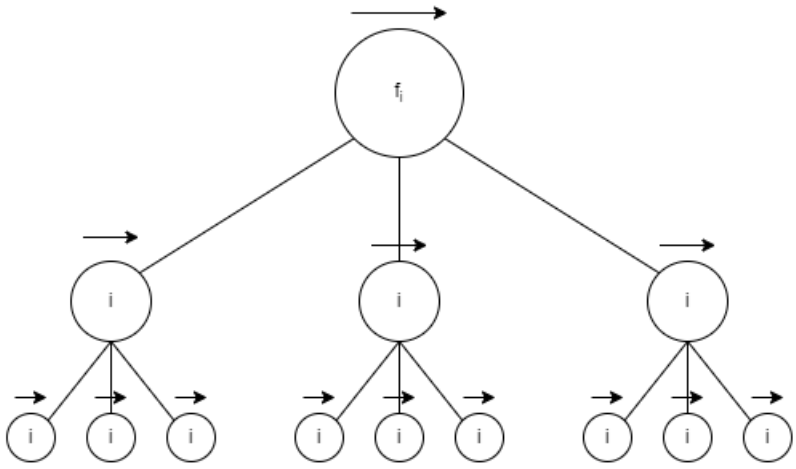


Figure 3.4. The nested data structures can be drawn as a hierachical structure of vectors. The idea is that we don't know the full embedding (see table for examples) and we would like to compute using the embeddings from each entity/object i.e. each i leaf in the model. The issue is how to infer the information from the embeddings upwards to gain f_i

mentioned in multiple key works. Those works is usually based upon learning a linear transformation matrix (such as in some of the works of mikolov et. al. [44]) typically notated as a translation matrix.

One approach is to align vector spaces whereafter a simple operation is possible. In such a case this is achieved with a rotation matrix.

Another approach is to do dimensionality reduction like concept et.al. did by doing PCA, SVM or similar where the objective is to learn the projection from k dimensions to k' dimensions that remove redundancy from concartanation.

3.2.4 Selected Embedding approaches

From investigating the state of the art in the area and aligning it with our data structure for SecDNS there are multiple approaches that was possible with embedding these nested categorical features which also include various approaches for encoding. Applying these embeddings schemers are validating them with the same simple classifier for best performance can help determine the preferred method to use.

The generalized approaches that was tried for the embedding are these structures:

1. 1) Flatten/collapse the dicts for a single representation as a string that can be embedded via word2vec, BERT or one-hot

2. 2a) Encode (one-hot, word2vec) each string and infer the embeddings onto each other via arithmetic/linear operations simple averaging , concatenation, weighted average, addition etc. Can be done in conjunction with embedding models for dimensionality reduction.

3. 2b) infer the embedding onto each other by feeding them to a feedforward NN after concatanate/averaging them to receive the collective embedding representation at the pooling layer. Requires normalization.

4. 2c) Build context vectors for categorical features such as the ones applied in doc2vec, cat2vec etc. to retain possible contextual information and effectively reduce dimensionality by using similarity.

5. 3) Map the features as graph and perform node2vec after node merging since that does not cosider hierachy or perform graph2vec for singular embedding representation

Generally, all of this methods can was applied along with variations of encoding, but the performance of them is dependent on the dataset in question and what is desired. Simple operations often proved useful but was more accurate when a context vector for the categorical features could be inferred as a weight through multiplication.

3.3 Machine learning techniques

Fundamentally, neural networks which are the foundation for deep learning are a set of algorithms designed to recognize patterns in a model. The networks works by interpreting sensory data through some mode of machine perception effectively labelling or clustering raw input. The output (i.e. recognized patterns) are numerical, contained in vectors into which all real-world data (images, sound, text, time series etc.) need to be translated. Neural networks can be viewed as a clustering and classification layer on top of the data stored/managed, helping to group unlabelled data according to similarities among the example inputs and classifying data when the network system have a labelled dataset to train. It should also be noted that neural networks can also extract features that are fed to other algorithms for clustering and classification which makes it possible to look upon deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression in context of implementations. Classification tasks depending upon labelled datasets is known as supervised learning i.e. humans transferring knowledge to neural network. Clustering (i.e. grouping) functions as the detection of similarities, are used for learning without the labels which is called unsupervised learning. With enough training deep learning is capable of establishing correlations between current events and future events effectively running regressions on the past and the future. Expanding this it is possible to introduce other algorithms to establish models working towards a goal i.e. reinforcement learning [45]. Technically, deep learning is stacked neural networks which is networks composed of several layers made of nodes where nodes is a place where a computation happens given a sufficient threshold is met for a condition. Each node combines input from data with a set of coefficients/weights which can amplify or damper it effectively assigning significance to input with regards to the task the algorithm is trying to learn which builds upon the network. The input-weight outcomes are summed and that sum is passed through nodes' activation function to determine whether or not the signal should progress further towards the final outcome. Essentially, input is fed to a input layer and progresses through a number of hidden layers of nodes (also called depth) to a final output layer. The feature extraction is often automated in such systems. In such networks concepts such as optimization algorithms such as gradient descent and activation functions such as softmax are methods to improve the complexity and accuracy of neural networks. To convert continuous signals into binary output a mechanism is the common logistic regression. In deep learning it is also advantageous to comprehend the concept of back-propagation. With this concept it is possible to efficiently compute the gradient of the loss function with respect to weights in a neural network in context of typical input-output scenario. That computation make it useful to utilize gradient methods for training multi-layer networks, updating weights to minimize loss commonly in concordance with the use of gradient descent or variants such as stochastic gradient descent. From a technical standpoint the back-propagation algorithm works by computing the gradient of the loss function

with respect to each weight by the chain rule iterating backwards layer-by-layer at sometime from the last layer to avoid redundant calculations of intermediate terms in the chain rule (e.g. dynamic programming) [46] [45].

Belief propagation in neural networks could be computed by firing rates of populations, which is multiple samples from probability distributions. The recurrent networks can integrate and fire neurons by performing belief propagation with spikes coding for sudden increases in probability. An advantage in neural networks belief propagation is the avoidance of inferring loopy and false beliefs by transferring only predication errors i.e. the difference inputs and a prediction of these inputs by the next layer or by a local inhibitory loops cancelling reverberated deviations [47].

Typically, Recurrent neural networks are used for malicious domain detection as it is compatible with real-time monotorization, however, some works have been using Convulotional neural networks to process text in context of natrual-language processing which is effective tools against malicious domains names [48].

Fundamentally, the reasoning for utilizing a neural network context for fulfilling the purposes of the SecDNS project lies in the advantages it presents. In context of the ERD graph in a neural network context the advantage is that it works in cases of unlabelled data which derives that no ground truth is necessary. It allows high level of uncertainty without the need of clustering which is computationally heavy. In case of RNN it is always updating the belief update and belief propagation. Another advantageous option with neural networks is the classification based on data features that exhibit a high degree of correlations to the core of the auxiliary malicious classification data feature. The main disadvantage is that neural networks are very dependent on the specificity of parameters which makes it difficult (and procedurally costly if desired to be corrected) to handle or change in response to changes (i.e. natural changes) often requiring re-training of the model.

3.3.1 Feed-forward Neural Networks

A Feedforward neural network (also known as multilayer perceptrons) have the goal to approximate function f^*. Essentially, to reiterate this goal for a classifier $y = f^*(x)$ maps an input x to a category y so in such a context the feedforward network defines a mapping $y = f(x; \theta)$ to learn the value of the parameters θ that results in the best function approximate. Such models have information flowing through the function being evaluated from x through the intermediate computations utilized to define f leading to the output y and in this context there are no feedback connections in which outputs of the model are fed back to itself [49] [50].

Works have been done in proposing neural networks that also propagates uncertainty about beliefs which is a is useful in malicious DNS detection as the beliefs are then updated according to uncertainty that is propagated through the network at eh introduction of current data [50].

3.3.2 Recurrent Neural Networks

The recurrent neural network is similar to the regular feedforward neural networks with the exception that there is a form of feedback connection. Essentially, connections between nodes form a directed graph along a temporal sequence (allowing dynamic temporal behaviour) by its use of the networks internal state (or memory) to process the sequences of inputs. RNN is classified into two broad classes with similar general structure i.e. finite pulses and infinite pulses. A finite impulse recurrent network is a directed acyclic graph which can be unrolled and replaced with a strictly feedforward neutral network whereas on the contrary an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled. Additionally, it is possible to have a stored state that may be under direct control of the network with recurrent neural networks which is also known as feedback neural network [46] [51].

As it relates to the SecDNS project, there have been several works (and patents) proposing neural network models trained for detecting malicious domains. Some works have utilized recurrent neural networks with ability to detect names generated by DGAs with high precision by being trained on a large set of domains generated by various malware [52]. In context of these large scale big data sets of DNS data a lot of methodologies that utilize apache spark clusters for storage and machine learning classifiers such as long short-term memory (LSTM) and deep learning approaches such as RNN networks since there have been proven to be accurate and quite scalable. In this context, domain names are encoded in two steps which is pre-processing by tokenization and dictionary creation of the training data set i.e. $input - data - shape * weights.of - character - embedding = (nb - words, character - embedding - dimension)$. where input $data - shape = (nb - words, dict - size)$, where $nb - words$ denotes the number of top characters, dict-size denotes the number of unique characters, each character is represented in on-hot encoding format and weight-of-character-embedding=(dict-size, character-embedding-dimension), character-embedding-dimension denoted the size of character embedding vector. The hyper parameter operation maps the discrete characters to its vectors which can then be passed through the layers in RNN. These systems are build upon some few known information such as features of URL, IPs, name length and other to establish a initial belief the model can then build upon [53].

3.3.3 Reinforcement Learning

In deep learning the concept of supervised learning (i.e. labelled data) and unsupervised learning (i.e. unlabelled data) is commonly used and known. Besides these two there is also reinforcement learning which is concerned with how software agents ought to take actions in a given environment in order to maximize a notion of cumulative reward. This form of learning does not necessarily need labelled data to be presented and bad actions does not need full corrections instead focusing on exploring new actions while retaining good actions i.e. exploring new knowledge while exploit-

ing current knowledge. Technically, this is typically formulated as a markov decision process (MDP). The markov decision process is in context a set of environment and agent states S, a set of actions A of the agents, the probability of transition from state s to state s' under action a is denoted as $P_a(s, s') = Pr(s_t + 1 = s' \mid s_t = s, a_t = a$, the immediate reward after transition from s to s' with action a denoted as $R_a(s, s')$, rules that describe what the agent observes which are often stochastic. A grade variety of methods for using current knowledge to determine which actions are good in a network in context of exploration have been proposed over the year e.g. cirterion of optimality by policy and state-value function, brute forcing, value functions, monte-carlo methods, temporal difference methods and direct policy search [49].

3.4 Explainable Artificial Intelligence

In its base principality, it can be said that Explainable Artificial Intelligence (XAI) is AI where the results of the intelligent solution is interpretable which is in contrast to the typical black box architectures followed by some machine learning algorithms especially neural networks. Machine learning has a long history of being applied to time-series data, however, the high complexity of those solution tends to make the interpretation difficult. Without a quantitative assessment it may be hard to understand areas of temporal data which may affect the prediction. Enter XAI which may offer a solution to these issues, adding the additional knowledge that may strengthen the overall model essentially aiding the models prediction by additional information [54] [55] [56].

In this PhD study we decided to focus on its application with time-series data as that is commonly used in NIDS systems and there are a multitude of algorithms to use when applying explainable AI on time series information.

The most common XAI algorithms in their base forms, namely SHAP (SHapley Additive exPlanations) and Class Activation Mapping (CAM) were studied closely. In addition, the study of the impact of well-known non-sequential based XAI algorithms called LIME (Local Interpretable Model-agnostic Explanations) was conducted to study the difference between sequential and non-sequential XAI algorithms in detecting malicious samples generated by adversaries.

Generally, the pipeline of applying XAI to a time-series would denote numerical values as a scalar from -1 to 1 signifying importance score or impact. To break this down simply we take the model and the raw data as input and obtain a vector of fixed length determined by a set sized sliding-window in which we train the model. The significance of the elements within what the vector consist of represent the impact the values at a given time-frame have on the prediction. So this numerical value represent whether a given step in the raw dataset affect the prediction negatively or positively, which is dependent on a baseline computed by using the model and raw data as input with the resulting prediction that can be used for comparison. This way XAI is applied holistically to the model and will explain what kind of importance areas of

the input stream has for a given feature which are values that can be used in a variety of ways such as enrichment or weight adjustment for the NN model. The properties of these values added possible enrichment as one of their inherent capabilities is that they are also affected by adversarial attack strategies which can make these values distinguishable dependent on what type of attack has been applied to them

For SHAPley we are calculating the impact an value has on its prediction (meaning it is independent from other categories). With an DL learner in time-series we calculate SHAP values by a window of values which can be aggregated together for a singular/shortened representation. The resulting vector is concatanated with values for each category feature so we end up with a series of vectors where each section of elements represent a feature explanatory value for a given time-frame. Furthermore, this value is a scalar from -1 to 1 that can be interpreted as an impact score, with the feature values given by a feature at time i, for the prediction of said feature derived from the pre-trained DL model if it took some baseline value. From the SHAPley software description the prediction starts from the baseline. The baseline for Shapley values is the average of all predictions. There is also other method of gaining the baseline such as an approximate method. Same concept can be applied to CAM version for time series data. LIME is a different story all together since it is a non-sequential model but the final output is still an impact score at each time-step which like SHAP and CAM can be summed for a singular representation.

In cases of univariate sqequential data it may be advantageous to just use the XAI values without applying any aggregation (some dimensionality reduction can be applied for very sparse cases) to retain the ordinal info and get a good representation of the impact (denoted as a score) over time. To that end, an sliding-window of a given size can be applied so we end up with a vector of a fixed size representing the XAI values for that chunk of samples. A lot of approaches aggregates these chunks to reduce the dimmensionality.

The concept is that by using the predictions and feature values we can gain a corresponding time of impact values for each feature at each time-step. These values are also affected by the adversarial strategies, therefore, enabling enrichment of our existing vectors of distance metrics between predictions and feature values with impact values which can all be distinguishable between legitimate and malicious samples. We essentially add in values that can be classified as malicious when compared to a legitimate sample and concatenate with our existing vector with the same property effectively creating more elaborate classifiers that possess more values that are distinguishable. This approach is somewhat novel but is based on what is already known about SHAP values and applying SHAP to time-series data which calculate Shapley values step-wise in a time series. This is also called permutation importance which is an approach that extracts importance per permutation enabled by XAI. It is worthy to note that SHAP and XAI in general are often used in context of multivariate models. The approach that was focused on in this study (and reflected in one of the included papers) is somewhat novel as it is based on single feature orderings, however, it conducted this single feature ordering done step-wise in frames of time samples within a time series making it effectively a combination of applying XAI

to time-serie by calculating impact values step-wise for each time-step applied to a single feature ordering (often called and approximate method with the TreeSHAP algorithm). The key goal of SHAP is to explain the prediction for any instance X_i as a sum of contributions from it's individual feature values which is then step-wise in time-series context. Effectively we end up with a sequential order of values ranging -1 to 1 which can be concatenated with our sequential order of distance metrics (between feature values and predictions) as an enrichment due to the explanatory values having the property of also being affected by adversarial samples making it possible to distinguish between malicious impact scores and legitimate impact scores. Same concepts are applicable for CAM, LIME or any other XAI representation.

3.4.1 SHAP (SHapley Additive exPlanations)

Shapley values can be summarized as the average of the marginal contributions across all permutations of a machine learning model. There are multiple advantages of SHAP. One of the advantages provided by SHAP include global interpretability which is the collective SHAP values showing how much each predictor contributes be it negative or positive to the target variable. The second benefit is the local interpretability where each observation gets its own set of shap values. Third the SHAP values can be calculated for tree based models [54].

Once the model has been run on with a dataset we can run the DeepExplainer and extract the SHAP importance score values locally [57]. So once we have a model from a collection of observations we can feed this into the explainer API for SHAP and extract some numerical values showcasing the importance of each feature. We can then use this to add weights to our re-trained network and thus gain a better model [58] [59].

3.4.2 Class Activation Mapping (CAM)

Unlike the SHAP which works best on a recurrent neural network model/ Long short-term memory the Class activation mapping (CAM) works best for Convulotional neural network model. With CAM we perform Global Average pooling on all features before the output layer for the CNN which is fed 1D information [60]. Those features which will be obtained are then fed to a fully connected layer with softmax activation which produce an output that is interpretable to our given framework i.e. an importance score. Unlike SHAP, the values for CAM are absolute so the importance of the features are determined in relation to one another which is a another step of calculation. Otherwise the input vector is the same i.e. vector of values [61]. The importance score like with SHAP are then used to retrain a more accurate model over time. Unlike with RNN and SHAP we do not need to fully intialize the model once before performing the explainer importance score extraction [54] [62] [63] [64] [65].

3.4.3 LIME (Local Interpretable Model-agnostic Explanations)

Another approach which we already took inspiration from with the SHAP approach is the non-sequential approach using LIME (Local interpretable model-agnostic explanations). LIME explains the prediction of any classifier in an interpretable manner by learning an interpretable model locally around the prediction. LIME does not consider the sequential order of data so a snapshot is sufficient. It is however more consistent. Otherwise, a similar pipeline with numerical importance scores was used [56].

3.5 Mobile Crowd Sourcing

One of the other major data sources that was utilized in multiple works in this study was data gained from mobile crowd sourcing (MCS). It was utilized since it is commonplace and readily available which makes the experimentation conducted easily replicable. Furthermore, it allows us to study the impact of adversarial machine learning for cybersecurity in a more general perspective. Pertaining to MCS, **System Model**: Where the actors involved in Mobile Crowd Sensing can be broadly classified as (i) *users*, (ii) *task initiators* and (iii) *infrastructure components* (Figure 3.5). Users are participants operating their mobile devices (e.g. smart-phones, tablets, smart vehicles), equipped with multiple embedded sensors and navigation modules. These platforms also possess transceivers that can communicate over wireless local area (i.e., 802.11a/b/g/n) and cellular networks (3rd Generation (3G) and Long Term Evolution (LTE)).

Task initiators (or campaign administrators) are defined as organizations, public authorities or, even, individuals that initiate targeted data collection campaigns, by recruiting users and distributing sensing tasks to them. Based on the adopted sensing approach, the recruitment depends on the desired degree of user involvement in the sensing process.

The back-end infrastructure is responsible for supporting the life-cycle of a sensing task. Some works suggest, or assume, the existence of a *centralized* infrastructure component to which users submit their data. Usually, such a centralized server is run by the campaign administrators so that they can have direct access to the results of the sensory tasks (i.e., reported sensor readings). As shown in Figure 3.5, the campaign administrator initiates sensing tasks by either uploading the corresponding specifications to a central repository (Step 1) or by directly distributing them to the targeted pool of users (Step 2). Users can then start participating in the sensing process and upload their contributions to the central server (Step 3); which may perform some type of aggregation on the submitted reports, depending on the application scenario. The collected (and possibly analyzed) sensor readings are finally released in various forms, e.g., attached to maps or as statistics, made available to the participants or to a larger public. Raw sensor data are collected on user devices and processed by local analytic algorithms towards producing consumable data for

requesting applications [66]. In this context, for a specific time window with n time steps, we consider a dataset D with the following format shown in (3.3),

$$D = \begin{bmatrix} v_{1,1} & \dots & v_{1,j} & \dots & v_{1,m} \\ \vdots & & \vdots & & \vdots \\ v_{i,1} & \dots & v_{i,j} & \dots & v_{i,m} \\ \vdots & & \vdots & & \vdots \\ v_{n,1} & \dots & v_{n,j} & \dots & v_{n,m} \end{bmatrix} \tag{3.3}$$

where $v_{i,j}$ reflects the value of sensor j at time step i. Therefore, we have a sequence of values, for each sensor, in the form of $S_j = [v_{1,j}, v_{2,j}, \cdots, v_{i,j}, \cdots, v_{n,j}]$, and this is where sequential analysis techniques, like LSTMs, are beneficial in extracting useful insights from the relationships between the various sequence values.

Threat Model: The aim of adversarial agents is to mislead the MCS applications towards considering malicious measurement values as legitimate in their services. To this end, an adversary may change several of the input values $v_{i,j}$ in S_j to $v'_{i,j}$, where $v'_{i,j} \neq v_{i,j}$, with the goal of maximizing the following constraint function:

$$\max f[x^t - x^{(a)}(\tau)] \qquad \text{s.t.} \quad f \leq T \tag{3.4}$$

where f denotes a utility function of the adversarial agent, x^t represents the actual ground truth measurement value and $x^{(a)}$ denotes the malicious measurement sample. The function can be linear or nonlinear and be more complex in formulation as in [19].

Looking at this model, there are two primary adversarial attack models [67]: 1) pre-training (poisoning) attacks, and 2) post-training (evasion) attacks. In pre-training attacks, adversaries try to inject malicious data in an attempt to poison the training dataset and, thus, decrease the classification accuracy of the classifier. In the post-training attack scenarios, adversaries aim at misleading trained classifiers to misclassify samples towards a malevolent intent.

Let us assume $f(x_i) = y_i$ as the mapping function to calculate/map x_i to y_i. For every new sensed values x'_i, f gives a new output $f(x'_i) = y'_i$, and we have the following cases:

- *True Positive*: if x'_i is positive and f correctly outputs positive, there is no loss on the application.
- *False Positive*: if x'_i is negative and f outputs positive, there is a loss ϵ on the application.
- *False Negative*: if x'_i is positive and f outputs negative, there is a loss l on the application.
- *True Negative*: if x'_i is negative and f correctly outputs negative, there is no loss on the application.

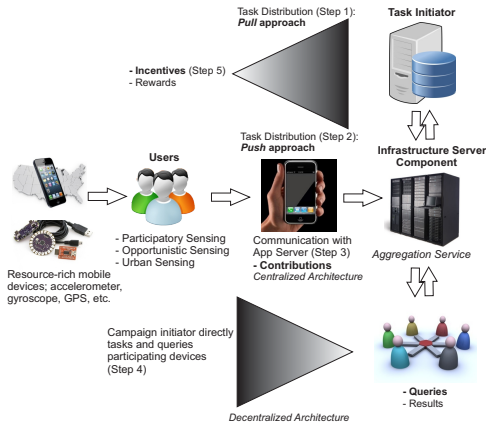


Figure 3.5. Architectural Overview of Mobile Crowd Sensing

In principle, a machine learning technique tries to minimize $|f(x'_i) - y'_i|$ which means minimizing l and ϵ. On the contrary, an adversarial attacker attempts to maximize the impact of the attack by maximizing $|f(x'_i) - y'_i|$. In the rest of the paper we refer to adversarial data as positive class of data, and legitimate data as negative class.

3.6 Belief Propagation in Ontologies

This section reviews the candidate approaches for propagating beliefs in the SecDNS project. Belief propagation is a broad topic in its own, but in the context in this study it referred to the challenge we were assigned to solve regarding how to effectively and correctly propagate and infer beliefs of malisciousness to domains. We also discuss limitations according to our concerns and our experiments.

3.6.1 Fundamentals of Belief

Belief is fundamentally viewed as the conviction of something being true. In cybersecurity environments such as the one for the SecDNS project it is about the belief of something being a malicious threat or not which is sometimes called benign. Conventional cybersecurity-by-design erects lines of defences within the computer/system infrastructure itself e.g. authentication mechanisms, firewalls etc. In data-driven cybersecurity which is the context that was relevant to this project, the beliefs is based on estimating the severity of threats based on observed data. This results in an estimation which forms a quantifiable belief from which final decisions can be motivated. What has been structured for beliefs in the project was a belief tuple $S = (\beta_1, ..., \beta_B)$

being a collection of $B \in \mathbb{N}$ possible beliefs that may be held about a problem. Consequentially, each constituent belief $\beta_{k\in\{1,...,B\}}$ therefore refers to a type of threat or vulnerability. Additionally, each belief can be assigned a numerical score b_k of severity, likelihood or risk where these score could be assembled into a belief vector denoted as $\vec{b} = \begin{bmatrix} b_1 \\ . \\ . \\ . \\ b_B \end{bmatrix}$ which shall satisfy $b_k \in [0,1], \forall k \in \{1,...,B\}$, and $\sum_{k=1}^{B} b_k = 1$ if $B > 1$, so as to live on a standard simplex $\beta = span\{\vec{b}\}$ which is referred to in this project as the belief space.

In the SecDNS project belief can be defined as a vector of $b_1, b_2, ..., b_n$. In application that means that for event E_i the totality of belief can be denoted as: $E_i(B) = b_1, b_2, ..., b_j$ regarding threats $t_1, t_2, ..., t_j$ derives that the classification of malicious entity/domain is based on the aggregation of set of belief B by threshold τ. The relationship of threats may be utilized to calculate the weights of beliefs in context of network applications of malicious domain detection.

3.6.2 Probability distribution

In the field of probability theory and statistics, a probability is a quantifiable measure of how probable it is for something to happen. Conversely, probability distribution is a mathematical function which provides the probabilities of occurrences of various possible outcomes in any given experimental scenario. Technically, the probability distribution is a description of a random phenomenon termed in probabilities of events. This probability distribution is specified by the underlying sample space the given experiment is build upon, which is the set of all possible outcomes of the random phenomenon being observed denoted by either real numbers, vectors and/or non-numerical values.

Generally speaking, when discussing probability distribution, it is divided into two classes, namely, discrete probability distribution or continuous probability distribution. The discrete probability distribution class is associated to scenarios where the set of possible outcomes is discrete values which mean that in this context it is possible for the distribution to be encoded by a discrete list of the probabilities of the outcomes which is also known as the probability mass function (PMF). In contrast, the continuous probability distribution class is applicable to scenarios where the set of possible outcomes can take values in continuous range such as real number, and is often described by probability density function (PDF) with the probability of any singular outcome actually being 0 which specifically encompass that a PDF must be integrated over an interval to yield a probability. It is noted that general probability measure (i.e. a real-valued function defined on a set of events in a probability space which must be assigned value 1 that satisfies measure properties) are often a necessity

in stochastic processes defined in continuous time such as can be typically be found in the field of deep learning [68].

In context of SecDNS, the probability distribution mainly concerns the possible outcomes of domains' maliciousness deemed by the classification of threats associated/believed about a domain i.e. the likelihood the domain variable assumes the possible values for maliciousness deemed by such things as IPs, query time, domain name etc. Both classes of probability distributions are relevant as both outcomes (e.g. IPS as discrete and query time as continuous) may be attributes used to do a summation of beliefs regarding a domains likelihood of maliciousness.

3.6.3 Probability Mass Functions

It is established that a probability mass function (PMF) is a function that gives the probability that a discrete random variable is exactly equal to some value. These values exists generally for either scalar or multivariate random variables whose domain is discrete. Formally, the PDF is the probability distribution of a discrete random variable, providing the possible values and their associated probabilities. Technically, it is the function probability $p : \mathbb{R}$ denoted as $px(x_i) = P(X = x_i)$ for $-\infty < x < \infty$. The probabilities that are associated with each possible values must be positive and sum to 1 and for all other values the probabilities need to be 0 denoted respectively as $\sum px(x_i) = 1$, $p(x_i) > 0$, $p(x) = 0$ for all other x. In this same domain there is an associate measure theoretic of these types of functions which view a probability mass function of a discrete random variable X as a special case of two more general measure theoretic construction namely the distribution of X and the PDF of X with respect to the counting measure [69].

3.6.4 Marginal Probabilities

In context of a lot of graphical models such as bayesian networks and neural network the concept of marginal probabilities is very present. In probability theory and statistics, the marginal distribution is defined in context of the probability of an event of one random variable irrespective of the outcome of another variable. Here, the marginal distribution (or marginal probability) is the probability of one event in the presence of all or a subset of outcomes of the other random variable. Consequentially, with that definition the marginal probability distribution is then defined as the marginal probability of one random variable in the presence of additional random variables. Fundamentally, given two variables the marginal probability is the sum/union over all the probabilities of all events for the second variable for a given fixed event for the fist variable. This means that marginal probability considers the union of all event's for the second variable rather than the probability of a single event which is present with conditional probability. Marginal probability is used advantageously in a lot of the works using bayesian models in the context of malicious domain detection discussed throughout this literature review documentation [70] [71].

In marginal context, the marginal probability mass function can be defined by given two discrete random variables X and Y whose joint distribution is known. Here, the marginal distribution of X is simply the probability distribution of X averaging over information about Y. Conversely, it is the probability distribution of X when the value of Y is not known consequentially calculated by summing the joint probability distribution over Y and vice versa which is denoted as equation:

$$p_X(x_i) = \sum_j p(x_i, y_i) \quad \text{and} \quad p_Y(y_j) = \sum_i p(x_i, y_j) \tag{3.5}$$

In the same context of marginal probabilities, a marginal probability can be written as an expected value: $p_X(x) = \int_y p_{X|Y}(x \mid y), p_Y(y)dy = E_Y[p_{X|Y}(x \mid y)]$. Typically in context of marginal probabilities the concept of conditional probability will be present in a comparative sense.

In the realm of probability theory, conditional probability is a measure of the probability of an event occurring given that another events has (by assumption) occurred which is formally written as if event of interest is A and the event is assumed to have occurred the conditional probability of B under the condition is $P(A \mid B)$ which should be recognizable by most who have had to deal with probability in some sense. In context of SecDNS the conditional probability could be inferred to the relationships between threats in relation to beliefs about domains i.e. calculating the probability of threat-1 (t_1) being present in domain-1 (d_1) if threat-2 (t_2) is known to being malicious for that domain.

As it relates to marginal probability with conditional probability in mind, the marginal probability of X is computed by examining the conditional probability of X given a particular value of Y then averaging this conditional probability over the distribution of all values Y where this follow the definition of expected value after applying the law of the unconscious statistician i.e.$\mathrm{E}_Y[f(Y)] = \int_y f(y)p_Y(y)\,\mathrm{d}y$. Resulting, the marginalization provides the rule for the transformation of the probability distribution of a random variable Y and another random variable $X = g(Y)$ denoted as: $p_X(x) = \int_y p_{X|Y}(x \mid y)\, p_Y(y)\,\mathrm{d}y = \int_y \delta\big(x - g(y)\big)\, p_Y(y)\,\mathrm{d}y$. [72].

In this marginal context it is also noteworthy to establish the marginal probability density function. This is determined by given two continuous random variables X and Y whose joint distribution is known then the marginal probability density function can be obtained by integrating the joint probability distribution over Y and vice versa given by: $f_X(x) = \int_c^d f(x, y)dy$, and $f_Y(y) = \int_a^b f(x, y)dx$ where $x \in [a, b]$, and $y \in [c, d]$. There are also marginal concepts for cumulative distribution functions and independence relationships of variables in this context [69] [72].

Summarily, in machine learning the likelihood of encountering multiple random variables is high. Obviously, machine learning was a key element of the SecDNS project. A random variable is fundamentally described as a variable whose values depend on outcomes of a random phenomenon which mathematically in context of

probability theory means that a random variable is understood as a measurable function defined on probability space whose outcome are typically numbers [73]. Assuming an relation of sorts between two variables there are three types of probabilities that are worth considering in the same context as marginal probabilities which will be found in plenty of works utilizing belief propagation in graphical networks (e.g. bayesian network) and neural networks. These probabilities are namely: joint probability which is the probability of events A and B, Marginal probability which is the probability of event $X = A$ given variable Y, conditional probability which is probability of event A given event B. As marginal probability is encountered a lot in these works about graphical model inference and neural networks as it relate to security it has been highlighted. Fundamentally, there is no special notation for the marginal probability as it is the sum or union over all probabilities of all events (by given two random variables) for the second variable for a given fixed event for the first variable formally denoted as $P(X = A) = \sum P(X = A, Y = y_i)$ for all y also known as the sum rule [70].

3.6.5 Markov Chains and Properties

Markov chains and properties are prevalent in a lot of bayesian network applications which is a common usage of belief propagation with works and techniques having been developed for the area of malicious domain detection relevant to SecDNS. The markov properties are found in a stochastic process if the conditional probability distribution of future states of the process which is conditional on both past and present values depends only upon the present state given that at present, the future does not depend on the past. It is defined by letting $(\Omega, \mathcal{F}, P)$ be a probability space with a filtration $(\mathcal{F}_s,\ s \in I)$ for some totally ordered index set I and let (S, S) be a measurable space where a (S, S)- valued stochastic process $X = \{X_t : \Omega \to S\}_{t \in I}$ adapted to the filtration is said to possess the markov property if, for each $A \in \mathcal{S}$ and each $s, t \in I$ with $s < t$, $P(X_t \in A \mid \mathcal{F}_s) = P(X_t \in A \mid X_s)$.. Here, in the case where S is a discrete set with the discrete sigma algebra and $I = \mathbb{N}$ this can be reformulated as $P(X_n = x_n \mid X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_n = x_n \mid X_{n-1} = x_{n-1})$. [74]. In context regarding neural networks, bayesian networks and other systems that uses belief propagation, the concept of markov chains keep coming up as it is a concept that is very useful in context of these methodologies. Fundamentally, markov chains is a stochastic model which describes a sequence of possible events in which the probability of each events depends only on the state attained in the previous events (or one state to another) where each state is a situation (or set of values). A collection of states are also known as a state space i.e. a list of possible states. The markov chain tells you the probability of hopping/transitioning from one state to any other state. So the Markov chain is a markov process (stochastic process that satisfies markov property) that has either a discrete state or a discrete index set (representing time). It is often defined as a process in either discrete or continuous time with a countable state space [75]. It is noted that there is also cohesive formal definitions

for each variation of markov chain be it discrete or continuous.

3.6.6 Belief Propagation Theory

In its base definition, belief is the estimation of something being true. In systems with multiple entities and associated data objects that can have assigned beliefs, it may be advantageous to utilize a methodology which combines these multiple beliefs to make weighted decisions. It is a necessity to utilize algorithms that propagate, aggregate and calculate belief in the context of neural networks and independently in context of scenarios such as heuristic systems or conventional machine learning.

The algorithm for propagating belief throughout a system known as belief propagation (also called sum-product message passing) is a message passing algorithm used to perform inference on graphical models by calculating the marginal distribution for each unobserved node/variable conditional on any observed nodes/variable. In the context of SecDNS the messages are events regarding data features which is beliefs about threats from malicious domains. It has been formulated as an exact inference algorithm on trees and in extension polytress. The base algorithm is defined as if $X = \{X_i\}$ is a set of discrete random variables with a joint mass function p, the marginal distribution of a single X_i is the summation of p over all other variables i.e.:

$$p_{X_i}(x_i) = \sum_{\mathbf{x}' : x'_i = x_i} p(\mathbf{x}'). \tag{3.6}$$

As it relates to malicious domain detection the belief propagation (BP) algorithm have been utilized in many works as other methods such as path-based inference algorithms are proven to be very expensive [76] [77]. With the purpose of increasing efficiency without too much loss a lot of approaches aim towards analyzing DNS data and establishing DNS graphs with great coverage through deep-analysis association [78] [79], signal utilization on graphs [80], belief rule based systems [81], graph mining [82], a variety of graph inference be it bayesian, markov random fields, max-produt BP (for optimization) [83] [84] [85] [86], feedforward neural networks [49], BP approximation [87], passive analysis [78] and more.

The belief propagation algorithm is a powerful tool that is utilized to help solve the inference problem that is found across multiple disciples including AI that is applied to a great variety of graphical models used to describe those issues. Some of the most common models that Belief propagation has a known application for include Bayesian networks, (pairwise) Markov random fields, the Potts and Ising models, Tanner graphs and factor graphs. We'll touch upon the most general and relevant in this review in relation to AI security applications but all is interesting topics to explore [88] [83].

3.6.7 Bayesian Inference and Graphical Networks

The key denotation for bayesian rules derives the posterior probability as a consequence of two antecedents i.e. a prior probability and a likelihood function derived from a statistical model for the observed data. The bayesian rules computes the posterior probability according to bayes' theorem: $P(H \mid E) = \frac{P(H|E) \cdot P(H)}{P(E)}$. Here the H is for hypothesis whose probability is affected by data, $P(H)$ the prior probability, E is for evidence, $P(H \mid E)$ posterior probability, $P(E \mid H)$ probability of observing E given H i.e. likelihood, $P(E)$ is marginal likelihood [89] [90] [91].

Formally, the definitions for bayesian inference are x a data point (possibly vector of values), θ the parameter of the data point's distribution i.e. $x \sim p(x \mid \theta)$ (vector of parameters), α, the hyperparameter of the parameter distribution i.e. $\theta \sim p(\theta \mid \alpha)$ can be a vector of hyperparameters, X is the sample a set of n observed data points (i.e. $x_1, ..., x_n$), $\tilde{x}$) a new data point whose distribution is to be predicted. In the context of bayesian inference the aspects of the prior distribution is the distribution of the parameters before any data is observed i.e. $p(\theta \mid \alpha)$, and for the sake of its determination it may be advantageous to use Jeffreys prior. The sampling distribution for bayesian inference is the distribution of the observed data conditional on its parameters i.e. $p(X \mid \theta)$ where this is termed as the likelihood that can be written in context as a function of parameters as $L(\theta \mid X) = p(X \mid \theta)$. For the marginal likelihood, which is also termed by the evidence is the distribution of the observed data marginalized over parameters i.e. $p(\mathbf{X} \mid \alpha) = \int p(\mathbf{X} \mid \theta) p(\theta \mid \alpha) \, \mathrm{d}\theta$. Determined by Baye's rules that is the foundation of Bayesian inference the posterior distribution which is the distribution of the parameters after taking into account the observed data [90] i.e.

$$p(\theta \mid \mathbf{X}, \alpha) = \frac{p(\theta, \mathbf{X}, \alpha)}{p(\mathbf{X}, \alpha)} = \frac{p(\mathbf{X} \mid \theta, \alpha) p(\theta, \alpha)}{p(\mathbf{X} \mid \alpha) p(\alpha)} = \frac{p(\mathbf{X} \mid \theta, \alpha) p(\theta \mid \alpha)}{p(\mathbf{X} \mid \alpha)} \propto p(\mathbf{X} \mid \theta, \alpha) p(\theta \mid \alpha) \tag{3.7}$$

Bayesian inference also lay out the basis for Bayesian prediction [89]. It is noted that works have been done showcasing how BP algorithm have been utilized and derived in context of bayesian inference application to compute the posterior distribution of the time evolution of the state of each given some observation [92].

As it relates to the SecDNS purposes of effectively doing malicious domain detection, bayesian inference is an effective tool in the classification problem with binary outcome. There have been works that used bayesian inference for the classification of domains which have a binary outcome where the aim is to classify a domain as either fast-flux or legitimate. In that context, bayesian classifier token (e.g. TTL) was used to correlate ASN's and A-record counts in order to calculate the probability of a domain being fast-flux or not. In a real-world perspective such methodologies have been used for email spam filtering by giving words found in emails probabilities of occurring in spam email and legitimate email. The probabilities are calculated

from training data set that has been manually classified as either spam or legitimate where once a classifier has been trained it is used for classification. The output is a likelihood total which can be used to mark domain as either fast-flux or not. The classifier for this DNS query response can be derived from bayes theorem as: $P(F \mid t) = \frac{P(t|F).P(F)}{P(t|F).P(F)+P(t|\neg F).P(\neg F)}$ where $P(F \mid t)$ is the probability that a domain is fast-flux if the token t is in the response, $P(F)$ is the overall probability that a domain is fast-flux, $P(t \mid F)$ is the probability that the token appears in a fast-flux domain query response, $P(\neg F)$ is the overall probability that a domain is not fast-flux, $P(t \mid \neg F)$ is the probability that the token appears in a legitimate domain query response [91] [93] [94].

Fundamentally, the reasoning for utilizing a bayesian network context for fulfilling the purposes of the SecDNS project lies in the advantages it presented. In context of the ERD graph in a bayesian network context the advantage is that both prior (initial) and posterior (belief) probability based on updating needs with ground truth is possible. However, the disadvantage present is the assumptions of a ground truth being present as this is a necessity. Another issue is that bayesian networks are relatively computationally heavy (due to clustering) in regards to the requirements that the data features for SecDNS have.

3.6.8 Marginal Probability in context of Graph Inference Applications

Progressively with the advent of data access there have been established big data sets of security information (network logs and application logs etc.) where efforts have been made to analyse it and create actionable security information. This detection problem can be viewed as a graph inference problem where a suggested solution was to adapt a workflow that construct a host-domain graph from proxy logs, seeding the graph with minimal ground truth information and using belief propagation to estimate the marginal probability of a domain being malicious. It is noted that the key to solving the inference problems lies in belief propagation in this context [86]. The methodology behind is a approach to detect malicious domains with the assumption that an enterprise's host-domain graph is present i.e. it is known that the domains are accessed by the enterprise's hosts. A graph is constructed by adding a node for each host in the enterprise and each domain is accessed by the hosts and then adding edges between a host and its accessed domains (build from multiple enterprise event log datasets such as HTTP proxy logs and DNS request logs). Another assumption in this approach is knowledge of a few nodes' states e.g. malicious or benign. The known nodes constitute a ground truth data set (the rest of the graph nodes are unknown nodes). By the given host-domain graph and the ground truth information established it can be constituted that the goal is to infer the states of the unknown domains in the graph. Formally, it is desirable to compute a node's marginal probability of being in a state i.e. the probability of the node being in a state given the states of other nodes in the graph. Here, marginal probability estimation is known to be NP-complete. In

this application, belief propagation is a fast and approximate technique to estimate marginal probabilities. The algorithm's time complexity and space complexity are linear in the number of edges in a graph which makes it scalable for large graphs making it an appropriate inference technique as it is quite scalable and successfully applicated in diverse fields including malicious domain detection. Technically, belief propagation relies on ground truth information and statistical dependencies between neighboring nodes to reliably estimate marginal probabilities where the dependencies are derived from domain knowledge [86] [95].

Denoting this technically, given an undirected graph $G = (V, E)$ where V is a set of n nodes and E is a set of edges, where every node is modelled $i \in V$ as a random variable denoted as x_i which can be in one of a finite set of states S constitutes the graphical model. The graphical model defines a joint probability distribution $P(x_1, x_2, ..., x_n)$ over G's nodes. Continued, the inference process computes the marginal probability distribution $P(x_i)$ for each random variable (denoted x_i). So in totality a nodes marginal probability is defined in terms of sums of the joint probability distribution over all possible states of all other nodes in the graph i.e. $P(x_i) = \sum_{x_1} .. \sum_{x_i - 1} \sum_{x_i + 1} .. \sum_{x_n} P(x_1, x_2, ..., x_n)$ where the number of terms in the sum is exponential in the number of nodes n. This is simplified with belief propagation which can approximate the marginal probability distributions of all nodes in time linear in the number of edges which is at most $O(n^2)$ complexity. So with the advantageous belief propagation algorithms it is possible to estimate a nodes marginal probability from prior knowledge about the graph's nodes and their statistical dependencies [86]. In this context, a nodes' (denoted i) belief (denoted $b_i(x_i)$) is i's marginal probability of being in the state x_i. The computation of $b_i(x_i)$ depends on priors of the graph nodes where a nodes' prior ($\phi_i(x_i)$) is i's initial (or prior) probability of being in the state x_i. The belief of the node is also dependent on the edge potential functions that model the statistical dependencies among neighboring nodes where the edge potential is denoted as $\psi_{ij}(x_i, x_j)$. The computation for the edge potential looks at two neighboring nodes i and j for the probability of i being in the state x_i and j being in the state x_j. By iterative message passing among neighboring nodes, the belief propagation achieves relative computational efficiency by organizing global marginal probability by smaller local computations for each node. So in the case of a node i and its neighbors $N(i)$ in each iteration of that algorithm i passes a message vector m_{ij} to each of its neighbors $j \in N(i)$ where each component $m_{ij}(x_j)$ of the message vector is proportional to i's perception j's likelihood of being in the state x_j. Summarily for this, the node's outgoing message vector to its neighbor j depends on i's incoming message vectors from its other neighbors and is computed by: $m_{ij}(x_j) = \sum_{x_i \in S} \phi_i(x_i) \psi_{ij}(x_i, x_j) \prod_k \in N(i) \backslash j) m_{ki}(x_i)$. The order of the message passing does not matter as long as all messages are eventually passed in each iteration. For the update which can be synchronous or asynchronous where with the advantageous synchronous it is possible to compute a node's belief values from i's incoming message in the converged graph or the last iteration: $b_i(x_i) = C\phi(x_i) \prod_{k \in N(i)} m_{ki}(x_i)$ where C is a normalization constant to ensure that

i's beliefs add up to 1 i.e. $\sum_{x_i \in S} b_i(x_i) = 1$. Additionally, several methods using tree approaches for belief propagation and score summation methods for belief propagation have also been proposed by several interesting contributors in the context of malicious domain detection as a graph inference issue [88] [86].

3.6.9 Belief Propagation for Bayesian Networks

Fundamentally, it is known that a bayesian network is a graphical model that represents the probabilistic relationships of a set of variable via a directed acyclic graph (DAG), where within variables are represented as nodes in bayesian network and their conditional dependencies are represented in directed edges. In this model each node is assigned an probability function of the variable. By using that dependency relation the network provide diagnostic (bottom-up, "symptoms to cause") and prognostic (top-down, "new information about causes to new beliefs about effects") probabilistic inference.

The general workflow that has been identified for some BP applications in Bayesian network (poly-tree with messages from parents to children and vice versa determined by the direction of edges) centers around formulas chronologically concerning likelihood (a product of all incoming messages sent by variable's children), priors (initial knowledge), belief (posterior probability), message to parents and message to children.

In such a network the inference can be made through the message passing algorithm introduced as Kim and Pearl's message passing algorithm (most frequently used for belief propagation), noting that this this typically works only on polytrees single connected networks, however, bayesian network has a DAG graph structure where two nodes have more than one path [77]. In cases where the multi-path connection has both direct (W-R) and indirect (W-VR) connections a method called clustering is utilized and developed to transform a graph into a probabilistically equivalent polytree as conventional message passing algorithm won't work in this scenario. The clustering method works by merging multiple nodes to remove the multiple paths between two nodes. Conversely, it is worth noting that the Junction Tree Clustering algorithm is used as a default algorithm for most major software packages for bayesian networks as it provides a systemic method of clustering consisting of three steps i.e. 1. Moralization where all parents and remove arrows are connected, 2. Triangulation where edges are added to ensure that every cycle has a length < 4, 3. Junction Tree where maximal cliques are identified to create supernodes. With such a transformation it is possible to use Kim and Pearl's message passing algorithm for updating beliefs. Here, the total strength of belief is a product of the casual support $\pi(X)$ from parent node and the diagnostic support $\lambda(X)$ from descendants denoted as $BEL(X) = \alpha\lambda(X)\pi(X)$ where a normalizing constant that makes $\sum_X BEL(X) = 1$. It ought to be noted that the node probabilities are associated with their conditional dependencies. There are two types of process methods for message passing called the top-down prognosis and the bottom-up prognosis. In the top-down prognosis the mes-

sage passing starts from calculating the belief of the root node where from the root the top-down message is propagated to leaves where the conventions $\pi\gamma(X)$ to which probabilities can be calculated using the conventional rules of probability calculation. Conversely, the bottom-up prognosis is the inverse of the top-down where message is passed up to the root [83] [77].

So, technically it can be stated that a bayesian netowrk is a directed acyclic graph of N variables x_i that defines a joint probability function :

$$p(x_1, x_2, ..., x_N) = \prod_{i=1}^{N} p(x_i \mid Par(x_i)) \tag{3.8}$$

Here, $Par(x_i)$ is the states of the parents of node i, and in scenarios where i has no parent the denotation $p(x_i \mid Par(x_i)) = p(x_i)$ is used. In relation, marginal probabilities are mathematically defined in terms of sums over all possible states of all other nodes in the system generalized for the last node $p(x_N)$ as the computation:

$$p(x_N) = \sum_{x_1} \sum_{x_2} ... \sum_{x_{N-1}} p(x_1, x_2, ..., x_N) \tag{3.9}$$

These approximate marginal probabilities are assigned as beliefs. For larger bayesian networks the belief propagation algorithm is appropriate [83].

3.6.10 Non-parametic Belief Propagation

To address the issue of applying conventional methods such as bayesian networks for belief propagation to models with continuous variables there has been works that extend belief propagation to continuous variables models by generalizing particle filtering and gaussian mixture filtering techniques for time series to more complex models resulting in a nonparametic belief propagation algorithm.

The workflow identified in works about non-parametic BP usually starts with the construction of a model their non-parametic representations. Thereafter there is established the Gibbs sampler for the product of several gaussian mixture, and then finally the application of the NBP algorithm. The NBP can simplified be broken down as given message it starts with determining the marginal influence to drawing the samples utilizing gibbs sampler and applied through conditional parameters it will be possible to construct message updates.

The non-parametic belief propagation algorithm (NBP) adresses the problem of analysing the messages in graphical models with continous (and non-gaussian) variables. The NBP achieves this by representing each message non-paramatically as a gaussian mixture. Performed exactly each message update produces a new product mixture with an exponentially larger number of components as NBP approximates this product mixture using an efficient Gibbs sampling procedure which allows a trade-off between statistical accuracy and computational efficiency. A derivation of interest

broke down the nonparametic belief propagation (NBP) algorithm for NBP update of nonparametic message $m_{ts}(x_s)$ sent from node t to node s. This technique is based on the belief propagation algorithm in graphs which are acyclic/tree-structure for the sake of calculating the desired conditional distributions $p(x_s \mid y)$ at iteration n of the algorithm where each node $t \in V$ calculates a message $m_{ts}^n(x_s)$ to be sent to each neighboring node $s \mid \Gamma(t)$ denoting the equation:

$$m_{ts}^n(x_s) = \alpha \int_{x_t} \psi_{s,t}(x_s, x_t)\psi_t(x_t, y_t) \times \prod_{u \in \Gamma(t) \setminus} m_{ut}^{n-1}(x_t)dx_t \tag{3.10}$$

where α is denoting an arbitrary proportionality constant. In the context of graphical models with continuous hidden variables, the resulting message nonparamtically is represented as a kernel density estimate. The kernel density estimation (KDE) is non-parametic estimate of the probability density function (PDF) of a random variable .Consequentially, this is derived by letting $\mathcal{N}(x; \mu, \lambda)$ denoting a normalized Gaussian density with mean μ and covariance λ evaluated at x. Hence, for the representation an M component mixture approximation of $m_{ts}(x_s)$ in the form of equation:

$$m_{t_s}(x_s) = \sum_{i=1}^{M} w_s^{(i)} \mathcal{N}(x_s; \mu_s^{(i)}, \lambda_s) \tag{3.11}$$

where $w_s^{(i)}$ is the weight associated with the i^{th} kernel mean $\mu_s^{(i)}$ and λ_s is a smoothing parameter and it is noted that the weights are normalized so that $\sum_{i=1}^{M} w_s^{(i)} = 1$. With these representations and a few assumptions about finiteness of the integrals it is possible to derive an updating BP algorithm for this non-paramatic context [96].

This type of belief propagation could have been suitable for the purposes of the SecDNS project as there is continuous information present (such as phishing) and distinct data (such as IPs) present in the project. Nonparametic belief propagation presents an option to handle continuous (typically non-Gaussian) distributions which may be desirable in context of interest rich high-dimensional variables as inference algorithms in graphical models for discrete approximation is infeasible in those contexts for continuous distributions.

3.6.11 Approximate Belief Propagation

Belief propagation can be used for general graphs (also called loopy belief propagation). In this context, all variables messages are initialized to 1 and updates all messages at every iteration. For the convergence of loopy belief propagation it is possible to do approximate visualization of the progress of belief propagation with techniques like EXIT charts in order to do an approximate test for convergence. Other methods for marginalization includes varied methods and monte carlo methods. An ordinary method for exact marginalization in general graphs called junction tree algorithm which is simply belief propagation on a modified graph guaranteed to be

a tree where the basic premise is to eliminate cycles by clustering them into single nodes [87].

3.6.12 Belief Propagation in Malicious DNS Graph Mining

Relating a BP utilization that relates to the purposes of malicious domain detection in the SecDNS project, it is noted that there have been some works done in the area of utilizing graph mining. A key beneficial aspect of a graph mining approach is that it takes the relationship between entities and events into consideration which can be inferred to the SecDNS project as it relates to the relationships between threats. Simplified this process establish a DNS graph composed of DNS nodes that represents server IPs, client IPs and queried domain in the process of DNS resolution to then applying graph mining task for malware detection by inferring graph node's reputation scores using the belief propagation algorithms where the nodes with lower reputation scores are inferred as those infected with malware at higher probability. The graph mining approach is used to do predictions based on the relationships of (domains,host answer ips) and (domain, ip clients). In those instances the BP algorithm functionality take a node on the graph only depending on properties of its adjacent nodes, which in malicious domain mining denote that some nodes are labeled as malicious (by using a labelled datasets) whilst the adjacent nodes interact with other via an inference function to update each other's beliefs. After that process the reputation scores are calculated by the BP algorithm. The base idea of the graph mining approach for malicious DNS detection is to step-wise build upon the graph building upon (d, IP) where d is the domain resolved to IP, develop algorithms for belief using belief propagation and markov random fields toward a novel proposal method for this security context. In context of DNS graph, the passive DNS data includes different aspects of DNS (A-records) which uses two columns formed as (d,IP) where d is domain which is resolved to IP. In that kind of DNS graph duplicates are removed from collected DNS queries visualising only connection between domain names and IPs. So in overview, the DNS gaph in this context is a bipartite graph with one side corresponding to domains and the opposite corresponds to IPs where an edge is formed from a domain d to an IP if there exists a record (d, IP). In context, a base reputation score (in scenarios not using summation) is assigned based on three given features namely network addresses, DNS zones and malicious clues. For the malicious clues there are different works for classification methods such as EXPOSURE that can be utilized where aspects such as query time, response content, TTL, and domain is included [82]. The ERD graph for SecDNS denotes similar entities. As it have been discussed about BP so far it is noted that an important feature of it in relation to markov random fields is the BP algorithm markov properties meaning that when a BP algorithm is in process on a graph where a message passing interacts between any two neighbors nodes. To understand that properly it a necessity to understand markov random fields in this context. Markov random field (MRF) is a set of random variables with markov properties described by an undirected graph. As discussed earlier in this

report an undirected graph is denoted as $G = (V, E)$ where $V = v_1, v_2, ..., v_n$ is the collection of vertices. In the context of markov random fields each vertex equals to a random variable because it has markov properties which means it only depends on properties of its adjacent nodes i.e. Given a node $v_i \in V$ and N_i being a neighbor set of nodes v_i if $(v_i, v_j) \in E$ and $v_j \in N_i$. Consequentially, MRF conforms to the local property: $P(v_i|v_{j_{v_j \in V/v_i}}) = P(v_i|v_{j_{v_j \in N_i}})$. As it relates to DNS the properties of this MRF model to DNS graph, each vertex of the DNS graph is corresponding to a random variable which represent the host IPs/Domain name. In order to detect malicious domain a knowledge set is defined as $D = (d_1, d_m)$ where $v_i = d_1$ indicating that the host IP/domain are malicious. The reputation scores for nodes and vertices (with part of vertex labels and relation knowledge) can then be calculated using a novel algorithm and mining algorithm respectively. One of the interesting proposal methods starts by assigning all nodes an initial value as malicious probability and marked as function $\phi(v_i)$. Here, the inference between neighbor is shown as function $\varphi_{ji} = P(v_i \mid v_j)$ which is the conditional probability for v_i that has a label v_i when node v_j has a label v_j. Due to the nature of an undirected graph the potential function of two neighbor nodes is symmetric, that is $\varphi_{ji}(v_i, v_j) = \varphi_{ji}(v_j, v_i)$ set by a value denotation that assign inference value addition or subtraction to legitimate or malicious by $0.5 + -\epsilon$ on the condition that $0 < \epsilon < 0.5$ determine the likelihood of a legitimate node being legitimate and vice versa for malicious where one presented implementation can assign $\epsilon = 0.01$. The key principle s that a message is sent from v_i to v_j to say how does node v_i think about the label of its neighbor where the message passing is marked as $m_{ij}(x_j)$. The calculation is for $\forall e_{ij} \in E$, the messages $m_{ij}(v_i)$ and $m_{ji}(v_j)$ are calculated as a follow function that is computing the message from node v_i to node v_j as: $m_{ij}^t(v_j) = \phi_i(v_i)\varphi_{ij}(v_i, v_j) \prod_{u \in N_i/j} n_{ui}^{t-1}(v_i)$. So when node v_i wants to send a message to node v_j it must collect all messages from it's neighbors before sending message to v_j. In context of the BP methodology, at any time all nodes send their link to the neighbors of them which is the case of this proposal is not necessary because not all nodes have knowledge to send. Therefore, the proposed condition is to only sent message if the malicious probability of a nodes is not equal to 0.5. Additionally, another rule/condition is that at each interaction the probability of knowledge node should not compute repetitively because the label of it is already known and it can't change regardless of what the neighboring nodes think. In BP methodology the summary of each node after an interaction is computed being in this context that it is the malicious probability for each of them (i.e. belief) denoted as $b_i(v_i)$. So, with N_i being the set of neighbor nodes of v_i and $m_{ui}^{t-1}(v_i)$ is all messages which v_i received in the last interaction a belief calculation is possible. This results in the belief calculation in each iteration of vertex being based on messages received from neighbors specifically:

$$b_i(v_i) = \frac{1}{Z_i}\phi_i(v_i)\varphi_{ij}(v_i, v_j) \prod_{u \in N_i/j} m_{ui}^{t-1}(v_i) \tag{3.12}$$

where Z_i is the normalization constant as the sum of beliefs on each label for

vertices should be 1. Combining the BP algorithms with the proposed conditions for neighbor N_i^* which is the neighbors of node i and the previous belief is not equal to 0.5 denotes formulas:

$$m_{ij}^t(v_j) = \phi_i(v_i)\varphi_{ij}(v_i, v_j) \prod_{u \in N_i^*/j} m_{ui}^{t-1})(v_i) \tag{3.13}$$

and

$$b_i(v_i) = \frac{1}{Z_i}\phi_i(v_i)\varphi_{ij}(v_i, v_j) \prod_{u \in N_i^*/j} m_{ui}^{t-1}(v_i) \tag{3.14}$$

So the message passing is operated by following these set conditions/rules and the initial beliefs sets which are given by a prior knowledge for some vertices to, then, for messages to be continuously iterated until the belief on a vertex is convergent or a limit for the iteration is reached for the algorithm [82] [88].

A proposal method offer some conditions to add on BP algorithm and can be applicable to malicious domain detection on DNS graphs where after a given initial value the reputation of graph nodes would be computed by belief propagation with some improvements which encompass removing nodes that don't connect in the established DNS graph (i.e. subgraphs with no knowledge is removed) and changing the probability of all nodes on the graph after receiving messages from neighbors where key idea is that the probability of a node will be aggregated from its neighbors while known malicious nodes are not changed. This approach achieved pretty good results for C and C [88]. Other interesting works that also utilizes the relation of markov random fields and belief propagation is work en stereo matching [95].

3.6.13 Considered Potential Belief Propagation (BP) Approaches

Throughout our research into the realm of belief propagation, we were able to formulate several potential approaches to be included in the broader context and framework of the SecDNS project (see Figure 3.6).

We divided belief propagation into two main approaches as the following:

- *Time Dependent BP*

  Time of the events is the most important factor in time dependent approaches, where the model updates the beliefs after each event chronologically. It is time dependent since an event updates the belief values of the involving entities right after the event. Therefore, if E is a sequence of events then we will have at least $|E|$ updates in our design matrices. As updating a belief value may trigger updating beliefs at other connected/related instances, we may have more updates due to propagation (more than $|E|$).

  Figure 3.7 depicts how we see events in a timeline where the current event may update the belief values of the involving instances. Time dependent approaches

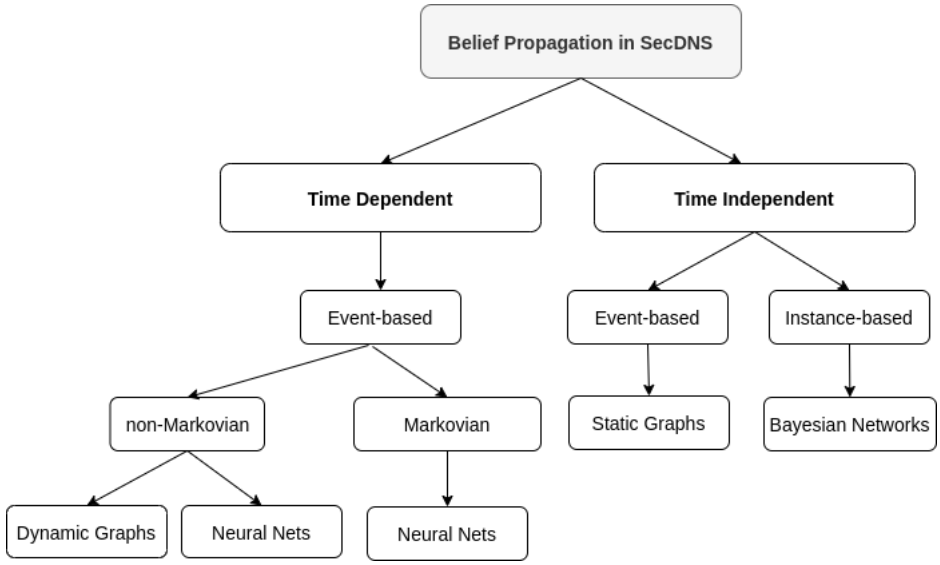


Figure 3.6. Potential Belief Propagation approaches in SecDNS

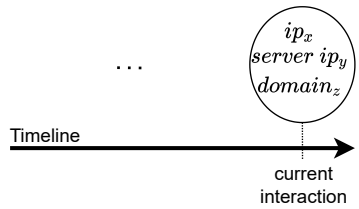


Figure 3.7. A time line and the current interaction between IP_x, a server with IP_y and $Domain_y$

propagate the maliciousness score (belief) of ip_x to update maliciousness belief of $domain_y$ and vice versa.

The event-based approach could be considered as Markovian or non-Markovian done sequentially or non-sequentially. Non-sequential techniques only consider the current event without looking into the history of the events related to the involving entities, while sequential based techniques attempts to find a pattern in historical events and updates.

- *Time Independent BP*

  This approach does not have to update the beliefs synchronously with the events. The updates may occur periodically by considering all events (Event-based), and the periods could be daily, weekly, etc. Moreover, the updates may follow an Instance-based approach which does not consider the events directly, but con-

siders the dependency between the entity types. Bayesian Networks that is a probabilistic graphical model can be an example for Instance-based approach which represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). In other words, instance-based approach attempts to build a general network to represent causality and dependability. In the context of SecDNS, we may calculate maliciousness probability of a domain (example.com) based on some observations and evidences (querying IP) which can be propagated through the network, modifying the probability distribution of other nodes that are probabilistically related to the evidence.

3.6.14 Synthetic Event-based Dataset

Event-based dataset contains collected events by a monitoring agent. Each event contains a list of the involving instances, type of the event, and a timestamp. Moreover, in event-based scenario beliefs are considered as feature of instances, so the monitoring agent tracks the features for each instance in form of matrices.

As access to real world dataset in SecDNS proved difficult to get, CSIS and DTU have collaborated to develop a synthetic data generation module (called Ontology) to simulate the events and the beliefs. The developed Ontology module requires a few parameters to be set for controlling the generation procedure. As each set of parameters generate an instance of an Ontology, so we can have different onthologies. Table 3.1 describes the parameters and the values that we set to generate different ontologies for our experiments in the following sections.

Table 3.1. Parameters used to generate synthetic datasets (ontologies)

Parameter	$Ontology_1$	$Ontology_2$	$Ontology_3$
Belief Propagation approach	Markovian	Markovian	Markovian
Symmetry	symmetric	symmetric	symmetric/asymmetric
Coupling	0.5	0.5	0.5
Interaction types	1	1	3
Entity types	1	2	3
Entity roles	2	2	7
Entity instances in each type	100	100	100
Intrinsic features	1	1	1
Extrinsic features	1	1	1
Train/Test ontology samples	7/3	7/3	7/3
Belief aggregation function(s)	mean	mean	mean
Timesteps	500	500	500

3.6.15 Markovian BP by applying Neural Networks

As we preferred simpler models with good performance level, we focused our simple neural network architecture (called BaseNN) before moving towards more complex neural network architectures like graph-based methods. Simpler ML models may have needed lower training time and they were easy to implement. Figure 3.8 shows the Markovian approach for the *DNSrequest* example shown in Figure 3.7. The model is Markovian as it only relies on the data (beliefs) from the current timestep when predicting new beliefs. The model predicts how input beliefs ($\vec{b}$) turn to output beliefs ($\vec{\beta}$) after an instance of the interaction. To compare our candidate methods for belief propagation, we used a Benchmark model (Figure 3.9) that simply forwards input beliefs as output beliefs ($\vec{\beta} = \vec{b}$). Once we have other candidate methods, we compared them to each other and also against the Benchmark model. This is also reflected in one of the included papers for this thesis.

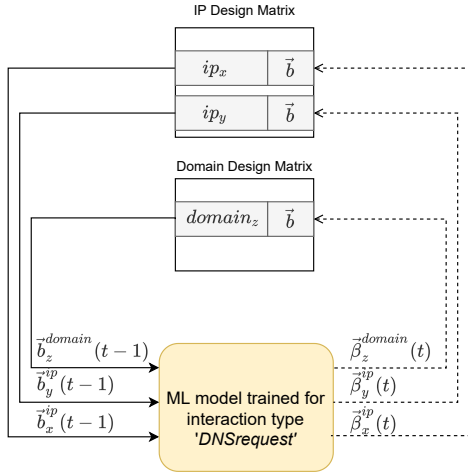


Figure 3.8. Markovian based Neural Network for the given example in Figure 3.7.

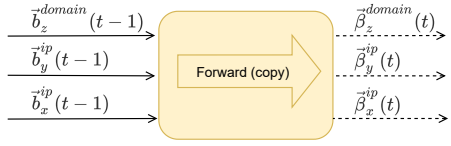


Figure 3.9. Benchmark model simply forwards (copies) input beliefs to output beliefs.

3.6.15.1 Evaluation of BP Framework

The most commonly used metrics in ML projects were used i.e. L1, Mean Square Error (MSE), Rooted Mean Squared Error (RMSE), and cosine[1] distance as our distance metrics. The distance metrics are calculated by equation where $\vec{b}_k^{(j)}(t)$ denotes real beliefs vector and $\vec{\beta}_k^{(j)}(t)$ is predicted beliefs vector of the entity instance k of type j at time t (equation 3.15):

$$D(\vec{\beta}_k^{(j)}(t), \vec{b}_k^{(j)}(t)) \tag{3.15}$$

If we assume the extrinsic features are identical to the the belief vectors (equation 3.16):

$$\vec{\chi}_k^{(j,i)} = \vec{\beta}_k^{(j)}(t) \tag{3.16}$$

then for any given interaction type i, we can quantify the performance of its learner with equation 3.17:

$$D^{(i)} = \frac{1}{L^{(i)}} \sum_{l}^{L^{(i)}} \frac{1}{\left|\xi^{(i)}\right|} \sum_{\forall_\epsilon \in \xi^{(i)}} D(\vec{\beta}_k^{(j)}(t), \vec{b}_k^{(j)}(t))c \tag{3.17}$$

where $\left|\xi_l^{(i)}\right|$ is the cardinality of $\xi_l^{(i)}$ and $L^{(i)}$ is the number of interaction instances of type i.

To make easier the comparison of different models, we calculate a disagreement percentage ($\delta_{x,y}^{(i)}$) according to equation (3.18) for each model trained on an interaction type, where $D^{(i)}$ is from equation 3.17 and $D_x^{(i)}$ means $D^{(i)}$ achieved by model x for interaction type i. $\delta_{x,y}^{(i)}$ shows how the models x and y are different in predicting the beliefs. Negative values of δ shows better performance for model y. In our evaluation, x and y are *BaseNN* and *Benchmark* models, respectively.

$$\delta_{x,y}^{(i)} = \frac{D_x^{(i)} - D_y^{(i)}}{D_y^{(i)}} \tag{3.18}$$

The outcomes evaluated with aforementioned metrics are mostly reflected in the included paper in this study: A proof of concept for supervised learning on ontologies. Using these metrics it was possible to draw conclusion with quantifiable proof, that stongly supported the viability of the Ontology Framework used in the SecDNS project which is central for this PhD study.

[1] Cosine similarity is a measure of similarity between two sequences of numbers. The *cosine distance* is defined as 1.0 minus the cosine similarity that is implemented in *sklearn.metrics.pairwise.cosine_distances* library.

3.7 Applying Theoretical Context Study

This chapter included the theory that was researched along with the formulated proposals and consideration that was gained throughout this PhD study which also encompassed certain implementations and experimentation that was evaluated. Hopefully, this theoretical context chapter provide a deeper knowledge base for the content of the included papers which is presented later in the thesis.

CHAPTER 4

Papers

The main contributions have been collected into 4 papers ready to/that have been published, also listed in the beginning of this thesis. The papers are included in their full format including their references. It is noted that at the time of submission of this thesis the papers "A proof of concept for supervised learning on ontologies" and "Explainable Artificial Intelligence to Enhance Data Trustworthiness in Crowd-Sensing Systems" are to be submitted, however, the versions included in this thesis are completed and will be (near) identical to the versions that will be submitted for publishing. Furthermore, associated papers to this thesis include An architecture for processing a dynamic heterogeneous information network of security intelligence which touches upon ontologies of similar constructs as the one used in the projects used in this thesis. Additionally, the source of DNS traffic was not the only data source used in this project. Mobile crowd sourcing data was also used in particular in the paper Fake sequential data detection. Regardless, the general concepts presented in these papers are more central to the overall objectives of what this thesis strive to achieve.

A proof of concept for supervised learning on ontologies

1st Amine Laghaout
Machine Learning Scientist
CSIS Security Group A/S)
Copenhagen K, Denmark
ala@csis.com

2nd Sajad Homayoun
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
sajho@dtu.dk

3rd Sam Afzal-Houshmand
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
saaf@dtu.dk

4rd Christian D. Jensen
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
cdje@dtu.dk

***Abstract*—Most applications of machine learning (ML) consist of scoring or labelling entities within a given problem domain (e.g., economics, medicine, cybersecurity, marketing, etc). Although the problem domain may comprise several entity types, most use cases of ML focus on scoring the instances of a single type only. Since different use cases may target different entity types, they often end up siloed from one another. For example, in cybersecurity, the labelling of Internet Protocol addresses (IPs) as either malicious or benign is a priori a different use case than labelling, say, domain names—thereby eliciting separate yet often redundant data-processing pipelines. More fundamentally, the fact that feature engineering is dictated by the use case overlooks the underlying relationships that are already built into the problem domain. For example, a domain name *is hosted at* an IP. Because of this ontological relationship, the scoring of one therefore correlates with the scoring of the other. For the sake of modularity, it is thus beneficial to systematically engineer such relationships into features independently of the use case at hand.**

In light of the above, we propose a framework which exploits the ontological structure of any problem domains so as to guide the feature engineering. In particular, we propose a reference architecture for systematically aggregating the features from entities that interact, i.e., whose properties change as a result of events they're involved in. We showcase our work with a fictitious ontology which demonstrates, at a high level, the predictive power of this feature aggregation.

***Index Terms*—belief propagation, machine learning, ontologies, cybersecurity**

CONTENTS

I. MACHINE LEARNING ON ONTOLOGIES: A MOTIVATION

A. Ontologies are the bigger picture

The field of machine learning has opened a floodgate of applications in countless problem domains that are based on data. Typically, the goal is to score or classify some entities that are of a specific type. Examples of such entity types and their corresponding problem domains are shown in Tab. I. The shortcoming of this customary approach—which centers the prediction on a single entity type—is that it fails to take advantage of the fact that predictions on that entity type are often determined by the broader context in which it evolves. This broader context is what is known as an *ontology*, i.e., the

Problem domain	Entity type	Prediction
personal finance	debtor loan	probability of defaulting on a load ROI for the creditor
economics	company stock	buy vs. sell
online advertisement	potential customer	clicks vs. doesn't click
medicine	images of tumors symptom	benign vs. malicious most likely disease
cybersecurity	file Uniform Resource Locator (URL)	legitimate vs. malware legitimate vs. phishing

TABLE I: This table shows examples of entity types from different problem domains, along with the possible predictions that can be made on them (i.e., regression or classification).

formal collection of entity types and relationships that make up the problem domain [1]. Let's illustrate this with the last example from Tab. I. For example, a URL is clearly not the only entity type that makes up the cybersecurity ontology. There are indeed many other entity types such as domain names, IPs, connected devices (e.g., computers, printers, etc.), and so on. Many of these other entity types are potential partakers in cybercrime. The probability that any given URL is involved in phishing[1] is therefore not solely based on its intrinsic properties such as its Hypertext Markup Language (HTML) content or its address string, but also on the reputation of other entities that are extrinsic to it but to which it is nonetheless related. These can be its parent domain, its hosting IPs, or even other URL instances it links to. Similarly for the example of whether one should buy or sell the stock of a given company, it may not be enough to consider the balance sheet of that one company. Instead, one should also factor in external entities such as the market share of competing companies, individual customers, or even—more abstractly—some general macroeconomic variables.

Considering the above, machine learning could greatly benefit from a framework that applies it to entire ontologies and not merely to a single homogeneous dataset where all training examples are independent and where the feature space is restricted to a single entity type. This article is an attempt to make some headway in this direction by proposing a systematic way to engineer features that capture the influence of the "environment" on an entity's score.

B. Scope

Note that our present proposal is simply a framework, i.e., a reference architecture for leveraging machine learning techniques on ontologies. Its goal is not to be faithful to any particular real-world ontology, since those tend to require tedious "handcrafting" of the datasets based on domain expertise. Instead, our ambition is to set the stage for realistic data by using simplified and controllable synthetic data. Given the ubiquity of ontologies, our framework should then be transposable to a wealth of problem domains.

The outline of this article is as follows. In §II, we give a qualitative tour of what ontologies consist of and then formalize their representation in a data-centric way. In particular, we spell out a high-level framework for (i) engineering features from any given ontology as well as (ii) subsequently running supervised learning on it. In §III, we showcase our framework with a simplified ontology of cybersecurity. In doing so, we shall mention the assumptions we have made in setting up such a proof-of-concept ontology. The results of our simulations are finally presented in §IV, and §V summarizes the main obstacles towards a more robust demonstration.

[1] "Phishing" is a type of cybercrime whereby the attacker creates a fake web page that looks like that of a legitimate business. The victims who land on that page thus run the risk of revealing sensitive information.

Fig. 1: Simplified ontology of microeconomics broken down into three entity types and seven interaction types. If the goal is to predict any company's stock price (†), then not only should the ML run on the company's intrinsic features (revenue, expenses, and historical stock prices), but it should also process features from whichever entities it interacts with. For example, it is reasonable to expect that the stock price of a company goes up whenever a new customer is acquired. Hence, the interaction `CustomerGained` is presumed to act as a (semi-deterministic) function which increases the score of the company instance $\texttt{company}_j$ it acts on based on the size of the customer instance $\texttt{customer}_k$. In general, whenever the company of interest undergoes an interaction, the ML should be able to predict the change in its stock price as a result of that interaction. Note how interactions are represented as functions whose arguments are *instances* of entities (hence the subscripts).

II. Ontologies: A data-centric representation

A. Ontologies as interacting entities

We shall define an ontology as a set of *attributed* and *heterogeneous* entities, whose attributes[2] are transformed in a well-defined way as they undergo *heterogeneous* interactions. An example of such an ontology is depicted in Fig. 1. Qualitatively, the characteristics of ontologies as per our definition above are as follows:

[2] I.e., features and targets.

1) Entities: An ontology is a collection of attributed entity types, also known formally as classes [1]. All entity instances of the same type live in the same feature space. It is therefore possible to stack the features of all entities of a same type into a design matrix. Fig. 1 shows diagramatically the design matrices of the three entity types. For example, the entity type `Company` has seven instances and three attributes, one of which is the target for supervised learning, namely the `stock price`. The design matrix for entity type `Company` is therefore 7×3. Since the ontology consists of heterogeneous entity types, one ends up with design matrices of different dimensions. As we shall see, unlike run-of-the-mill supervised ML, training a model on ontologies cannot depend on a single design matrix, but should instead aggregate features from disparate design matrices.

2) Interactions: The dynamics of an ontology are driven by what we shall call interactions (or events). From an information-theoretic perspective, it is the interactions that transform the attributes of the entities over time. For our purposes, these interactions are mathematical functions that transform the attributes of the entities that they involve. The goal of ML in this context should therefore be to learn the function behind each interaction so as to predict its effect on the entities involved. In other words, in the bigger picture of ontologies, the role of ML cannot limit itself to predicting the scores of the entities as if they existed independently of each other. Instead, it should capture the effect of their environment—i.e., the other entities that are interacted with—on those scores. The prediction on any given entity is therefore based not only on its intrinsic features, but also on its extrinsic features, i.e., on the abstract embeddings that summarize the history of interactions it took part in. This idea will be formalized in §II-D.

B. The state of the art and its limitations

It is tempting to see many similarities between the problem of ML on ontologies and the field of graph machine learning (GML) [2]–[4]. A priori, there may seem to be a one-to-one mapping between vertices and entities on the one hand and edges and interactions on the other; see, for example Fig. 5 for a tentative entity-relationship diagram of cyberspace. Many applications of GML to ontologies such as social networks [5], pharmacology [6], [7], or even cybersecurity [8], [9] have indeed proven feasible. However, these applications are not general enough to capture the ontologies in the way we have defined them in §II-A. In particular, to our knowledge, none of the off-the-shelf GML frameworks currently available (e.g., StellarGraph [10], GraphSage [11], etc) can handle entities that are

1) attributed,
2) heterogeneous, and
3) interacting over time.

An even more fundamental limitation of GML is that graphs are by default based on edges, namely *bipartite* relationships. Interactions, on the other hand, can in principle be multipartite and hence not necessarily decomposable as pairwise relationships.

C. Notation for the main concepts

Let's now formally represent ontologies in a machine-readable way. As could already be seen from Fig. 1, all information about the heterogeneous, attributed entities can be stored in design matrices. As is typical of all (supervised) ML datasets, these design matrices are made up of the targets and *intrinsic* features. The latter are referred to as intrinsic because they only depend on the entity types they're attributed to. However, as we've already argued, one also needs extra information to summarize the history of interactions that the entity underwent. This, we claim, can be represented by an extension of the feature space with *extrinsic* features. At any given time t, the resulting anatomy of the data for an entity instance k of type j, $\varepsilon_k^{(j)}$, is depicted in Fig. 2 where

- $\vec{b}_k^{(j)}(t)$ is the vector of the target features,
- $\hat{f}_k^{(j)}(t)$ is the vector of intrinsic features, and
- $\hat{\chi}_k^{(j,i)}(t)$ is the vector of extrinsic features resulting from interactions of type i. Several such interaction types may exist, each involving a dictionary $\xi^{(i)}$ of entity types keyed on their role in the interaction. It is these extrinsic features which characterize ML on ontologies since they're the ones that propagate the influence of the interacting entities among each other.

The entity $\varepsilon_k^{(j)}$ is therefore fully represented by the data vector that concatenates ($\oplus$) the above vectors:

$$\hat{d}_k^{(j)}(t) = \overbrace{\vec{b}_k^{(j)}(t)}^{\text{targets}} \oplus \overbrace{\hat{f}_k^{(j)}(t)}^{\text{intrinsic features}} \oplus \left[\overbrace{\bigoplus_{\forall i | \varepsilon^{(j)} \in \xi^{(i)}} \hat{\chi}_k^{(j,i)}(t)}^{\text{extrinsic features}} \right] . \quad (1)$$

Notice that the extrinsic features are themselves concatenated for each interaction which involves entity type j. Furthermore, while $\vec{b}_k^{(j)}$ and $\hat{f}_k^{(j)}$ have well-defined dimensions that are dictated by the domain knowledge (and the availability of data), the dimension of $\hat{\chi}_k^{(j)}$ is in fact a hyperparameter since it is a model-dependent embedding of the interactions.[3] This is because the extrinsic "features" are not givens but are instead to be inferred. In the parlance of deep learning, $\hat{\chi}_k^{(j,i)}(t)$ is therefore more akin to a hidden layer than an input layer, except that—unlike typical hidden layers—the extrinsic features are persisted in the design matrices of the entities and can thus be regarded as an extension of the feature space.

D. A framework for supervised learning

Given the features, the goal of supervised learning is to infer the targets, i.e., the collection of scalars $\vec{b}$. This means that over all $K^{(j)}$ entities of type j, one needs to learn the blackbox

[3] In the simulations of §IV, we shall simplify $\hat{\chi}_k^{(j,i)}$ to be a scalar aggregated from the one-dimensional targets of the neighboring entities.

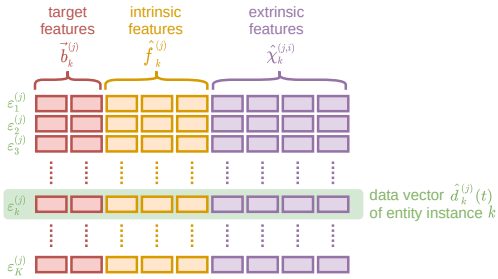


Fig. 2: Notation for the design matrix of entities of type j. The novelty with machine learning on ontologies is that one needs to engineer the so-called extrinsic features $\hat{\chi}^{(j,i)}$ which, unlike the intrinsic ones, aren't present *ab initio*, but should instead be elicited from the sequences of each interaction of type i undergone by the entities. Note that, for simplicity, we've only shown the extrinsic features resulting from interactions of type i. In principle, there could exist several interaction types that involve entities of type j.

function $\mathcal{M}^{(j)}$ that acts on the features so as to approximate $\vec{b}_k^{(j)}$ with

$$\beta_k^{(j)} = \mathcal{M}^{(j)}\left(\hat{f}_k^{(j)}(t) \oplus \left[\bigoplus_{\forall i | \varepsilon^{(j)} \in \xi^{(i)}} \hat{\chi}_k^{(j,i)}(t)\right]\right). \quad (2)$$

We shall abstract away the ML model which emulates $\mathcal{M}^{(j)}$ so long as its loss function is a distance metric[4] $D\left(\vec{b}_k^{(j)}, \vec{\beta}_k^{(j)}\right)$ between the actual and inferred target vectors. In other words, the optimization over $\mathcal{M}^{(j)}$ is

$$\underset{\mathcal{M}^{(j)}}{\operatorname{argmin}} \frac{1}{K^{(j)}} \sum_{k=1}^{K^{(j)}} D\left(\vec{b}_k^{(j)}(t), \vec{\beta}_k^{(j)}(t)\right). \quad (3)$$

E. Engineering of extrinsic features

We have up to now been putting off the explanation of how $\hat{\chi}_k^{(j,i)}$ is formed from the data vectors of the interacting entities. In other words, we still haven't specified how the interactions behave as mathematical functions. Let's denote such a function for interaction type i as the black box $\mathcal{M}_\chi^{(j,i)}$. It takes the data vectors of the interacting entities as inputs and returns the updated extrinsic features of $\varepsilon_k^{(j)}$ as outputs, i.e.,

$$\mathcal{M}_\chi^{(j,i)} : \left\{\vec{d}_k^{(l)}(t-1)\right\}_{\forall \varepsilon_k^{(l)} \in \xi^{(i)}} \to \hat{\chi}_k^{(j,i)}(t) \quad (4)$$

where the curly braces encompass the data vectors from each element in the set $\xi^{(i)}$ of entities involved in interaction type i. For simplicity, we assume that the interaction took place at the discrete time step $t-1$. As mentioned earlier, the extrinsic features aren't available from the outset, but are instead aggregated "on the fly" as the interactions unfold. I.e., whenever an interaction happens, one should update the relevant entries in the design matrices of the entities that interacted.

By substituting (4) into (2), we get

$$\beta_k^{(j)} = \mathcal{M}^{(j)}\left(\hat{f}_k^{(j)}(t) \oplus \left[\bigoplus_{\forall i | \varepsilon^{(j)} \in \xi^{(i)}} \mathcal{M}_\chi^{(j,i)}\left(\left\{\vec{d}_{k'}^{(l)}(t-1)\right\}_{\forall \varepsilon_{k'}^{(l)} \in \xi^{(i)}}\right)\right]\right). \quad (5)$$

It therefore transpires that the optimization in (3) pertains both to $\mathcal{M}^{(j)}$ and to $\mathcal{M}_\chi^{(j,i)}$. Whereas $\mathcal{M}^{(j)}$, is the overarching mapping of the features onto the targets $\vec{b}_k^{(j)}$, $\mathcal{M}_\chi^{(j,i)}$ can be thought of as a latent layer that is transfer-learned from any given interaction type i.[5] The model architecture that reflects (II-E) is depicted in Fig. 3. (For a more in-depth presentation of this entire framework, see [13].)

III. A case study in cybersecurity

A. Cyberspace as an ontology

We shall now showcase our proposal with a semi-realistic ontology, namely one which approximates the problem domain of cyberspace. Our goal shall be to score the entities of cyberspace with a probability of being involved in a malicious activity (e.g., phishing, ransomware, spyware, etc). For example, if the entities of interest are IPs, the intelligence thus generated would be stated in plain English as, say,

> some IP `154.108.17.20` is believed to be involved in a malicious activity with probability[6] $\vec{\beta}^{(\text{IP})}_{154.108.17.20} = 0.03$.

Similar statements about the probability of maliciousness could in principle be made for all entities that make up the ontology of cyberspace. A tentative representation of such an ontology is given by the entity-relationship diagram in Fig. 5. Note that this shows the entity and interaction *types*. Their *instances*, as they unfold in time, would instead appear in a time-stamped event log such as that of Fig. 6.

B. The need for synthetic data

Ideally, the data sets that could be used to showcase our framework should be both

(i) labelled (i.e., the target vectors $\vec{b}$ should be provided for all time steps prior to t), and
(ii) dense (i.e., all intrinsic features $\hat{f}$ should be provided at all times).

In cybersecurity, this implies that one can monitor the activities of all entities in cyberspace and that, up to and including time step $t-1$, all such entities are labeled with

[4] Here again, the choice of the particular metric is left to the end user.

[5] Indeed, the engineering of $\hat{\chi}_k^{(j,i)}(t)$ could be thought of as a case of transfer learning [12].

[6] We shall refer to the vector of targets $\vec{\beta}$ as the *belief* vector. This terminology is inherited from the subfield of *belief propagation* [15] and its wide use in cybersecurity, particularly in the context of trust management [16].

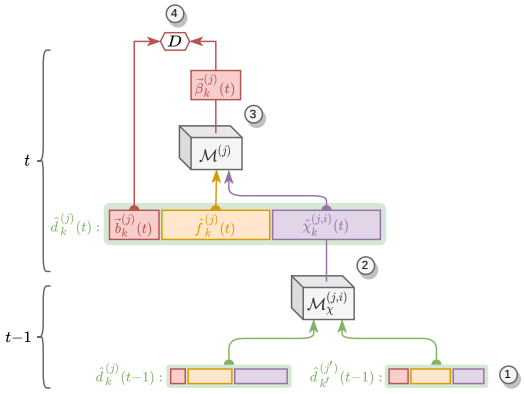


Fig. 3: Computational graph for training ML on ontologies. This depicts the two trainable modules required to predict the targets $\vec{b}_k^{(j)}$ of any entity $\varepsilon_k^{(j)}$ of type j at some time step t. The entity is shown to have undergone an interaction at the previous time step $t-1$ together with some other entity $\varepsilon_{k'}^{(j')}$ of type j' ①. The vector embedding that summarizes the effect of this interaction on $\varepsilon_k^{(j)}$ is generated by the trainable module $\mathcal{M}_\chi^{(j,i)}$ ②. This embedding $\hat{\chi}_k^{(j,i)}$ is concatenated with the intrinsic features $\hat{f}_j^{(j)}$ to then be processed by the final trainable module $\mathcal{M}^{(j)}$ which in turn outputs a prediction $\vec{\beta}_k^{(j)}$ of the targets ③. The training is driven by a distance metric D between the actual and predicted targets, which doubles as a loss function, and that is calculated at the very end ④. Notice how this architecture is somewhat reminiscent of wide & deep learning [14], whereby the wide and deep parts correspond to $\mathcal{M}^{(j)}$ and $\mathcal{M}_\chi^{(j,i)}$, respectively.

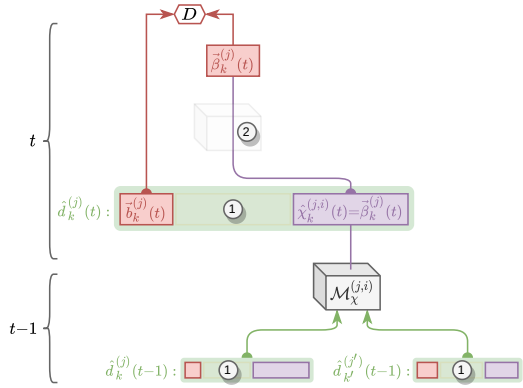


Fig. 4: Computational graph based on two simplifying assumptions. The first is that there are no intrinsic features; i.e., only the targets get propagated between the different entities ①. The second is that the predicted targets are identical to the extrinsic features ②. In other words, $\mathcal{M}^{(j)}$ is an identity operator and can thus be omitted from the computational graph. The learning on the ontology is therefore reduced to learning the functions $\mathcal{M}_\chi^{(j,i)}$ for the individual interactions. (Here again, to minimize clutter, we are only showing a single, bipartite interaction.)

a ground truth $\vec{b}$. Such an omniscience over any ontology is unlikely. In practice, the data sets would be much more sparse and the availability of ground truth much more limited. This said, the very potential of applying ML on ontologies would still hold true, albeit with a much worse performance. For the purpose of demonstrating our framework, we shall therefore use a synthetic ontology which approximates that of cyberspace.

C. A systematic approach to the simulations

To be able to bring more flexibility in simulating an environment, the introduced ontology requires a set of input parameters (ontology parameters). Running the ontology with different parameters gives us the freedom to study various environments with different complexity. For example, we may want to study how the number of entity types affects the complexity of an ontology and subsequently the performance of the target learning mechanisms (e.g. machine learning models). One may simulate a simple ontology with only one entity type and one interaction type, or may go with more complex environments by defining more entity/interaction types. Table II describes each parameter.

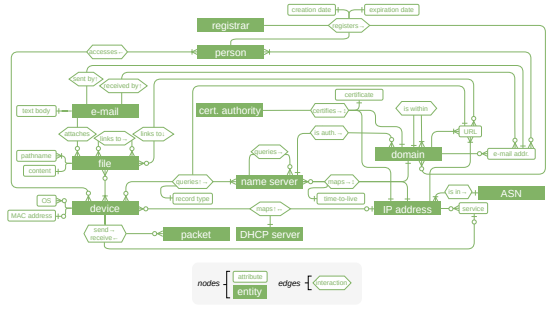


Fig. 5: Ontology of cyberspace represented as as an entity relationship diagram (ERD). The nodes are either entities (green background) or attributes thereof (white background) whereas the edges (diamonds) are interactions—or more generally, relationships—between two or more nodes. Note that this ERD represents entity and interaction *types*.

Timestamp : **Event**

2019-07-24 16:27:32:01 : DEVICE{my-desktop} **queries** NAME_SERVER{pns23.cloudns.net} for RECORD_TYPE{A} of URL{http://www.csis.dk/}

2019-07-24 16:27:32:49 : NAME SERVER{pns23.cloudns.net} **maps** DOMAIN{csis.dk} to IP_ADDRESS{94.130.180.46}

2019-07-24 16:27:33:00 : DEVICE{my-desktop} **receives** PACKET{PcktID#9784-a} from SERVICE{port:80} of IP_ADDRESS{94.130.180.46}

2019-07-24 17:30:07:13 : E-MAIL{ID#56789} **sent by** E-MAIL_ADDR.{john.doe@csis.dk} and **received by** E-MAIL_ADDR.{jeff.roe@csis.dk} **attaches** FILE{Win32bot.exe}

Fig. 6: Time-stamped event log. Each event consists of interactions (ERD edges in **bold**) between entities (in SMALL CAPS). The name of the entity instances is mentioned within the curly braces.

TABLE II: Describing setup parameters.

parameter	description
number of int. types	the number of ways that the entities interact with other entities. The definition of each interaction type can be added by adding function types such as mean aggregation to the ontology.
symmetry	determines how the interactions between entities affect their extrinsic features and target features. *Symmetric* means all involving entities in an interaction receives updates with the same aggregation function, meaning the same values for the target and extrinsic features. *Asymmetric* means the involving entities gets different levels of update. Note that this parameters applies on all interaction types.
coupling	shows the degree of impact from the history of inputs when an interaction types is calculating the updates to extrinsic/target features. Coupling can be in $[0, 1]$, where 0 means the data from the past has no impact in updating the target features.
number of time-steps	the duration of simulation shown in number of time steps. As there is only one interaction instance for each time step, so it also shows the total number of interaction instances.
number of entity types	the total number of entity types. Each interaction type involves a single or multiple entity types.

D. Assumptions and pitfalls

For this case study we assume that the interactions follows the Markovian process. It means that for any given time, the conditional distribution of future states (beliefs) depends only on the present state and not at all on the past states. In our case study we assume the target features (maliciousness belief) of the entities involving in an interaction instance would be updated based on their maliciousness at the time of interaction. We also assume only the target features get propagated between the different entities, which means no intrinsic features. Figure 4 depicts a simplified version of the computational graph given in Figure 3, where $\mathcal{M}^{(j)}$ is dropped and the loss function D would be calculated using the output from $\mathcal{M}_{\chi}^{(j,i)}$ and the real target features $\bar{b}_k^{(j)}(t)$. We also assume the propagation of targets are only based on the immediate neighboring entities meaning that there is only one extrinsic feature. More extrinsic features can be added by any distance.

By simplifying the ontology, we may face plateaus as a potential challenge here where the outputs converges to a certain value after a significant number of interaction instances between different entity instances. the calculated targets features converge to a certain value for all entity instances after a significant number of interaction instances which can be different between ontologies with different complexity. To avoid plateaus issue, one may cut simulation after detecting plateaus based on *moving average* of target features. For example, it could be a sign of plateaus if the moving average of the target features for most entity instances has not been changed significantly for the last w timesteps.

E. Pipeline for supervised learning

Figure 7 depicts the underpinnings of our pipeline for training regression models to learn from history of changes in target features (beliefs) of the involved entities for each interaction type of our case study. After learning from the history, we will be able to predict new values of target features after a certain interaction instance. The design matrices and interactions are the input to the pipeline and the output would be a regression model per each interaction type. Therefore, we will be able to predict the impact of any interaction instance using the regression models.

The training phase comprises two steps: (1) data wrangling and preprocessing and (2) training the regressors. Data wrangling step receives the ontology containing all interactions and design matrices, and creates a train-test dataset by concatenating (equation 6) all target features from the entities involved in each interaction instance. The ontology contains target feature values for both before and after the interaction instance. In other words, the interaction instance based on a certain rational spits out output vector (V_{out}) after feeding the input vectors (V_{in}). For example, an interaction instance outputs the mean value of the input target features of the involving entities instances. Therefore, for each time step we have input vectors and output vectors showing before and after the interaction instance, respectively.

$$V_{in} = \bigoplus_{\forall \varepsilon_k^{(l)} \in \xi^{(x)}} b_k^{(l)}(t) \tag{6}$$

The step for training the regressors outputs a single trained regressor for each interaction type, which is able to predict

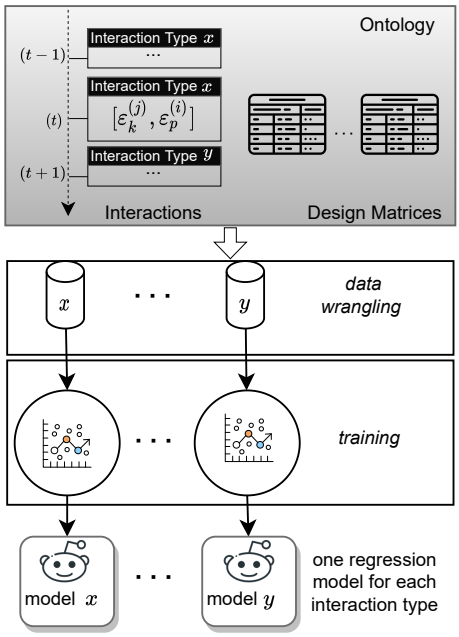


Fig. 7: The pipeline to train a ML model for each interaction type. There is a single regression model for each interaction type (x, y etc. in this case) trained from separate dataset for each interaction type prepared by the *data wrangling* step. Note that the input to the data wrangling step could be multiple ontologies, so the regression models have been trained using the data from different runs of the same environment setup (parameter setup).

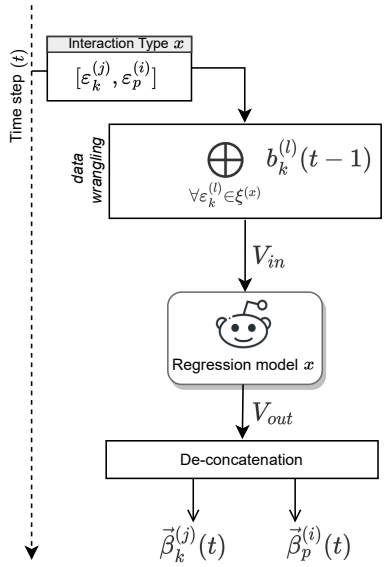


Fig. 8: Belief forecasting for an interaction instance of type x at time step t.

output vectors based on input vectors fed to the same interaction type. We use regression based algorithms in this case study, as we assumed the target features are a score showing maliciousness score of an entity instance. One may want to use classification if the target features are categorical features.

After the regressors are trained and are ready to predict the output vectors, we can use the testing pipeline shown in Figure 8 in an online manner. Assume we have an interaction instance of type x as $[\varepsilon_k^{(j)}, \varepsilon_p^{(i)}]$ at testing time step t. Then, there is a data wrangling step to prepare the input vector by concatenating all input target features (equation 6) of each involving entity instances. Based on the type interaction, x in this case, the input vector (V_{in}) is fed to the certain regression $model\ x$ for predicting output vectors (V_{out}). Finally, there is the de-concatenation step (reverse of equation 6) that separates the target features from the output vector. By this step we have a single belief for each entity.

IV. Results

The results section consist of a description of the experimental setup, the evaluation metrics used and the performance of various ontologies fed to the different learning methods.

A. Experiments

To apply the pipeline for supervised learning, we needed to set up a few environments to study how different parameters impacts the complexity the environment, and subsequently the performance of the supervised learning. We limited our study to the environments given by the combination of parameter values shown in Table III. However, it is possible to easily simulate environments with higher level of complexity by considering higher number of interaction/entity types etc. The combination of different values in Table III make 9 environments which is given by multiplying count of all possible values for all parameters. As interaction types could be either symmetric or asymmetric, we can design our experiments as Table IV by considering interaction symmetry. It worth noting that in environments with a single interaction type, having both symmetric and asymmetric interaction types at the same time is invalid as a single interaction type could be of type either symmetric or asymmetric (not both). Therefore, in the table we do not see a row with one interaction type and Symmetry of *Sym-Asym*. A simple ontology may only have one interaction type and one entity type, while an ontology may simulate a more complex environment with 3 interaction types and 3 entity types. In our case study, experiment 1 and 24 are the simplest and the most complex environments, respectively.

B. Parameter Description

For the experiments it is mainly the parameters of interaction types, entity types and symmetry that are focused on

TABLE III: List of values assigned to each parameters

parameter	value(s)
number of int. types	$1, 2, 3$
number of entity types	$1, 2, 3$

Exp. no.	# of Ents	# of Ints	I1	I2	I3	Symmetry
1	1	1	S	-	-	Sym
2	1	1	A	-	-	Asym
3	1	2	S	S	-	Sym
4	1	2	A	A	-	Asym
5	1	2	S	A	-	Sym-Asym
6	1	3	S	S	S	Sym
7	1	3	A	A	A	Asym
8	1	3	A	A	S	Sym-Asym
9	2	1	S	-	-	Sym
10	2	1	A	-	-	Asym
11	2	2	S	S	-	Sym
12	2	2	A	A	-	Asym
13	2	2	S	A	-	Sym-Asym
14	2	3	S	S	S	Sym
15	2	3	A	A	A	Asym
16	2	3	A	A	S	Sym-Asym
17	3	1	S	-	-	Sym
18	3	1	A	-	-	Asym
19	3	2	S	S	-	Sym
20	3	2	A	A	-	Asym
21	3	2	S	A	-	Sym-Asym
22	3	3	S	S	S	Sym
26	3	3	A	A	A	Asym
24	3	3	A	S	A	Sym-Asym

TABLE IV: The constitution of ontologies denoted by the degrees of freedom in number of entity types, number of interaction types and symmetry (i.e. symmetric, asymmetric or symmetric-asymmetric)

since they are kept variable, thus, functioning as the degrees of freedom of the experiments. Therefore, it is necessary to have clear definitions of what these parameters constitute in the context of the framework for our case study Hence, the definitions of the parameters used in implementing the framework are stated.

1) Entities: For the entities there are 3 main classes that have been defined which is *ClientIP*, *ServerIP* and *DomainName*. Per definition when we denote 1, 2 or 3 entity types it refers to these 3 types. So with 1 interaction type we have chosen ClientIP, and with 2 entity types we refer to the ClientIP types and ServerIP being present. Given by the defined the symmetry the score of infection is assigned and will be affecting designations of both types i.e. if there is an interaction between entity of type ClientIP and ServerIP the score is calculated with an aggregate function (i.e. *mean*, *max* etc.). Same applies with 3 entity types where the DomainName entity is introduced. These entity types are assigned randomly but generated to make equal amount of the generated dataset.

2) Interactions: The interaction types we are using in the experiment is defined as I_1, I_2 and I_3, where I_1 and I_2 are bipartite interactions (between two entities), and I_3 is a tripartite interaction type consisting of three entities. Downloading a file from a server could be an example of bipartite interaction types between ClientIP and ServerIP, while DNSRequest could be an example of tripartite interaction where it connects three entity types consist of ClientIP, ServerIP and DomainName.

Beliefs ascribed to entities can change after an interaction i.e. interactions can be considered a function that takes input beliefs and derive output beliefs. The interaction is Symmetric if the impact is the same for all involving entities, for example, the output will be *mean* of the inputs (other options could have been maximum, median, minimum etc.). Whereas if the interaction uses different aggregation functions for computing output beliefs, such as a combination of mean for one entity and maximum for the other entity, it is an Asymmetric interaction type. There is situations with multiple interaction types where one type could be symmetric and the other is asymmetric which is given the distinction of being symmetric-asymmetric. Furthermore, in tripartite the interactions the interaction types can be symmetric using the same aggregate functions (e.g. mean) for all three entities and asymmetric when a different aggregation method (e.g. mean, median and maximum) for all three entities.

3) Symmetry: In the experiment level, symmetry is a reference to the whole experiment based on the containing interactions. If all interactions are symmetric meaning that the aggregation method is the same, e.g. mean, max or median, on all interacting entities the experiment symmetry is considered Symmetric. Contrarily, if the interactions are asymmetric meaning that the aggregation method differs for each interacting entity. With experiments that have multiple interaction types there is a possibility that one type could be symmetric while every other type is asymmetric (and vice versa) resulting in the experiment being considered Symmetric-Asymmetric. To generalize this lets consider the real beliefs input for two entities as b_1 and b_2. The predicted beliefs of each of those interacting entities (denoted as $b_1^{'}$ and $b_2^{'}$) are calculated by an aggregate function F, where if the the aggregate functions are equivalent we consider the interactions symmetric and if not we consider them asymmetric, which can be stated as:

$$\begin{aligned} b_1^{'} &= F_1(b_1, b_2) \\ b_2^{'} &= F_2(b_1, b_2) \end{aligned} \tag{7}$$

where $F_1 = F_2 \leftrightarrow$ symmetric, and $F_1 \neq F_2 \leftrightarrow$ asymmetric.

4) Experimental Setup: To follow a fair performance evaluation, we decided to run each ontology 10 times, so we had 10 files including interactions and design matrices. We then divided the files into 9 and 1 files for training and testing, respectively.

To show the performance of our supervised learning pipeline, we selected five common regression algorithms namely *k-nearest neighbors regressor (kNNR)*, *decision tree s (DTR)*, *random forest regressor (RFR)*, *deep neural networks (DNNs)*. We used 5-fold cross-validation to evaluate and compare the performance of our trained models. After finding the best trained model of each algorithm, we used our validation ontology to evaluate how well the best model of each algorithms predicts the updated beliefs for each interaction type. We compare the performance of the supervised framework

with a benchmark model which simply forwards the input beliefs as the output beliefs without any further process. It worth noting that we removed the plateaus from validation data to avoid biasing the evaluation towards simple *Benchmark* model.

C. Evaluation metrics

The distance metric is calculated by equation where $\vec{b}_k^{(j)}(t)$ denotes real beliefs vector and $\vec{\beta}_k^{(j)}(t)$ is the predicted beliefs vector of the entity instance k of type j at time t (equation 8):

$$D(\vec{\beta}_k^{(j)}(t), \vec{b}_k^{(j)}(t)) \tag{8}$$

where D could be L1, MSE[7], RMSE[8] or any other metrics for evaluating the distance between predicted and real target values.

If we assume the extrinsic features are identical to the target features (equation 9):

$$\vec{\chi}_k^{(j,i)} = \vec{\beta}_k^{(j)}(t) \tag{9}$$

then for any given interaction type x, we can quantify the performance of its learner with equation 10:

$$D^{(x)} = \frac{1}{L^{(x)}} \sum_{l}^{L^{(x)}} \frac{1}{\left|\xi^{(x)}\right|} \sum_{\forall \varepsilon_k^{(j)} \in \xi^{(x)}} D(\vec{\beta}_k^{(j)}(t), \vec{b}_k^{(j)}(t)) \tag{10}$$

where $\left|\xi_l^{(x)}\right|$ is the cardinality of $\xi_l^{(x)}$ and $L^{(x)}$ is the number of interaction instances of type x.

D. Performance

For the 24 experiments of differing ontologies based on variable parameters the measured expectation and outcomes was that the more complex ontologies (i.e. higher number of types and asymmetric) would denote a worse performance overall across learning methods if the ontology framework is functioning correctly. To give a better overview it has been decided to include a table for each metric across experiments as rows (1-24) and methods (kNNR, DNN, Benchmark, DTR, RFR) as columns. As a point of reference the experiments number indicate the combination of number of entities, interactions and symmetry constitute the ontology for said experiment.

Tables V compare the performance of each studied method using RMSE metric as our D metric in equation (8). The results show that all learning metrics have achieved better performance in comparison to the benchmark technique (BM). The tables show that the performance of the studied models are much better for simpler environments with less entity/interaction types, and this verifies the idea behind the defined parameters in ontology framework. For example, we expect more complex dataset when increasing the number of entity or interaction types, and this complexity could be proved by comparing the performance of the learners. This can be proved by comparing RMSE of experiment 24, which is the most complex, with other experiments.

Exp.	BM	kNNR	DNN	DTR	RFR
1	0.0793	0.0024	0.0171	0.0051	0.0019
2	0.0807	0.0024	0.0335	0.0051	0.002
3	0.0922	0.0028	0.0279	0.0072	0.002
4	0.0926	0.003	0.0293	0.0072	0.002
5	0.0859	0.0034	0.0222	0.0083	0.0035
6	0.0642	0.0038	0.0235	0.0082	0.0048
7	0.0829	0.0051	0.0433	0.0102	0.0051
8	0.0805	0.0046	0.0356	0.0093	0.0053
9	0.25	0.0028	0.033	0.0093	0.0053
10	0.1103	0.0005	0.0331	0.0011	0.0007
11	0.1076	0.0013	0.0683	0.0022	0.0015
12	0.1176	0.0016	0.0531	0.0031	0.0018
13	0.105	0.0012	0.0593	0.0025	0.0016
14	0.1402	0.0005	0.0511	0.0014	0.001
15	0.1176	0.0016	0.0433	0.0031	0.0018
16	0.1013	0.0035	0.0284	0.0069	0.0033
17	0.1088	0.0016	0.0284	0.0044	0.0029
18	0.25	0.0032	0.0371	0.0046	0.0033
19	0.25	0.0061	0.0654	0.0095	0.0059
20	0.2278	0.0061	0.0315	0.0113	0.0062
21	0.1279	0.0008	0.0675	0.0017	0.001
22	0.249	0.0045	0.0374	0.0056	0.0047
23	0.25	0.0443	0.0291	0.0495	0.0444
24	0.1302	0.0031	0.0067	0.004	0.0033

TABLE V: The comparative RMSE metric across experiments and learning methods

V. Conclusion and outlook

The outcomes of the experimentation of the framework showcase indication that observable deviances can be detected when increasing the complexity of an ontology as presented in this paper. Generally, this confirms that a generated ontology can effectively mimic real-life appliances.

Furthermore, we have observed the effectiveness of various ML techniques including deep learning as compared to a benchmark giving high indication that the application of such techniques are extremely viable in the context of detecting malicious sources based on DNS data.

Acknowledgment

This work was supported by the Innovation Fund Denmark (IFD) under the SecDNS project (grant number 8090-00050B).

References

[1] *Ontology in Computer Science*. London: Springer London, 2007, pp. 17–34. [Online]. Available: https://doi.org/10.1007/978-1-84628-710-7_2

[2] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," 2020.

[3] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[4] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," 2018.

[5] S. Al-Eidi, Y. Chen, O. Darwishand, and A. M. S. Alfosool, "Time-ordered bipartite graph for spatio-temporal social network analysis," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, 2020, pp. 833–838.

[7] Mean Squared Error

[8] Rooted Mean Squared Error

[6] T. Gaudelet, B. Day, A. R. Jamasb, J. Soman, C. Regep, G. Liu, J. B. R. Hayter, R. Vickers, C. Roberts, J. Tang, D. Roblin, T. L. Blundell, M. M. Bronstein, and J. P. Taylor-King, "Utilizing graph machine learning within drug discovery and development," *Briefings in Bioinformatics*, vol. 22, no. 6, 05 2021, bbab159. [Online]. Available: https://doi.org/10.1093/bib/bbab159

[7] R. Mercado, T. Rastemo, E. Lindelöf, G. Klambauer, O. Engkvist, H. Chen, and E. J. Bjerrum, "Graph networks for molecular design," *Machine Learning: Science and Technology*, vol. 2, no. 2, p. 025023, mar 2021. [Online]. Available: https://doi.org/10.1088/2632-2153/abcf91

[8] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, ser. CIKM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2077–2085. [Online]. Available: https://doi.org/10.1145/3269206.3272010

[9] X. Sun, M. Tong, and J. Yang, "Hindom: A robust malicious domain detection system based on heterogeneous information network with transductive classification," 2019.

[10] C. Data61, "Stellargraph machine learning library," https://github.com/stellargraph/stellargraph, 2018.

[11] J. Leskovec, "Graphsage: Inductive representation learning on large graphs," https://snap.stanford.edu/graphsage/, 2017.

[12] S. Bozinovski, "Reminder of the first paper on transfer learning in neural networks, 1976," *Informatica*, vol. 44, pp. 291–309, 2020. [Online]. Available: https://doi.org/10.31449/inf.v44i3.2828

[13] A. Laghaout, "Supervised learning on heterogeneous, attributed entities interacting over time," *CoRR*, vol. abs/2007.11455, 2020. [Online]. Available: https://arxiv.org/abs/2007.11455

[14] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, ser. DLRS 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 7–10. [Online]. Available: https://doi.org/10.1145/2988450.2988454

[15] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proceedings of the Second AAAI Conference on Artificial Intelligence*, ser. AAAI'82. AAAI Press, 1982, p. 133–136.

[16] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996, pp. 164–173.

A Perfect Match: Deep Learning Towards Enhanced Data Trustworthiness in Crowd-Sensing Systems

Sam Afzal-Houshmand*, Sajad Homayoun*, Thanassis Giannetsos‡
*Technical University of Denmark (DTU), Cyber Security Section, Denmark
‡Ubitech Ltd., Digital Security & Trusted Computing Group, Greece
Email: saaf@dtu.dk, sajho@dtu.dk, agiannetsos@ubitech.eu

***Abstract*—The advent of IoT edge devices has enabled the collection of rich datasets, as part of Mobile Crowd Sensing (MCS), which has emerged as a key enabler for a wide gamut of safety-critical applications ranging from traffic control, environmental monitoring to assistive healthcare. Despite the clear advantages that such unprecedented quantity of data brings forth, it is also subject to inherent data trustworthiness challenges due to factors such as malevolent input and faulty sensors. Compounding this issue, there has been a plethora of proposed solutions, based on the use of traditional machine learning algorithms, towards assessing and sifting faulty data without any assumption on the trustworthiness of their source. However, there are still a number of open issues: how to cope with the presence of strong, colluding adversaries while at the same time efficiently managing this high influx of incoming user data. In this work, we meet these challenges by proposing the hybrid use of Deep Learning schemes (i.e., LSTMs) and conventional Machine Learning classifiers (i.e. One-Class Classifiers) for detecting and filtering out false data points. We provide a prototype implementation coupled with a detailed performance evaluation under various (attack) scenarios, employing both real and synthetic datasets. Our results showcase how the proposed solution outperforms various existing resilient aggregation and outlier detection schemes.**

***Index Terms*—Mobile Crowd Sensing, Adversarial Machine Learning, Data Trustworthiness, LSTM, One-Class Classifier**

I. Introduction

The emergence of resource-rich devices has changed the landscape of mobile sensing with multiple embedded sensors, e.g., accelerometers, cameras, etc. With more than 6 billion mobile subscriptions worldwide, these sensors (collectively) can be used to sense the environment and gather valuable data of unprecedented quality and quantity, practically from everywhere. This new sensing paradigm, *Mobile Crowd Sensing* (MCS) [1], makes individuals and user communities the focal point of the sensing infrastructure.

Despite all benefits of MCS, its applications still face the critical challenges of data quality and integrity of sensed data [2] and could suffer from incorrect contributions due to their inherent open nature; Over the past decade machine learning has been among the top methods to gain hidden insights from IoT data. ML systems are trained using datasets that are assumed to be *representative* and *trustworthy* whilst malicious actors can impact the decision-making algorithms by either targeting the training data (poisoning attacks) or forcing the model to their desired output, e.g., misclassification of abnormal events (evasion attacks).

In this context, security problems have already been addressed in the form of Adversarial Machine Learning (ML). There are many proposed solutions leveraging conventional ML-based techniques towards separating malicious data from benign. Game theory has also been exploited in the design of convolutional neural networks to detect tampering [3]. Concept drift, which is a common phenomenon in IoT data, has also been considered in problems such as feature extraction [4].

However, one main hurdle in all such mechanisms is how well they can operate in a distributed environment, like the one entailed in MCS, comprising many untrustworthy data sources providing falsified data (either as the result of an attack or a malfunctioning sensor). Compounding this issue, Banerjee et. al [5] studied how concept drifts and false data can masquerade the existence of intelligent attackers and their impact on the accuracy of both the clustering and (unsupervised) classification processes; however, they did not investigate the integration of more advanced Deep Learning schemes. What is needed is *a set of advanced deep learning mechanisms for assessing and sifting faulty data without any assumption on the trustworthiness of their sources while coping with intelligent adversaries that try to significantly decrease the overall performance, cause targeted misclassification*.

This paper meets this challenge with FSD - a novel data verification framework employing deep learning techniques to address the issue of possible data poisoning in MCS environments. We define malicious data as falsified data exhibiting different statistical properties (from the real data provided by benign users), and our approach is leveraging sequential relationships of sensory data towards detecting malicious samples generated by adversaries or faulty components. More specifically, our solution named False Sequential data Detection (FSD) offers: (i) Data verification by combining Deep Learning sequential architecture of Long Short Term Memories (LSTMs) with conventional one-class classifiers for distinguishing between "false" and "real" samples, (ii) Proof-of-concept implementation evaluated under various testing scenarios using both real and synthetic datasets in order to have more flexibility on the type of experiments. FSD demonstrates high accuracy even when the distribution of data comes from adversaries that demonstrate very similar behavior with the legitimate user data; i.e, strong colluding adversaries by composing two main attack strategies that represent different aspects of adversarial machine learning.

II. Related Work

A problem that is inherent when applying machine learning solutions for false data detection in MCS infrastructures is the concept-drift that occurs in some types of sensors used in emerging applications such as environmental monitoring; e.g., temperature and humidity [5]. Most recent studies, however, mainly focus on challenges in the context of sensing task allocation, sparse sensing, privacy, and data integrity [6]. But only a few works try to address the problem of false data detection [7], [8]. For instance, there are data verification techniques such as Deco [8] which uses spatio-temporal techniques to reconstruct missing values. Deco, however, suffers from high deviations. DETECT-and-CORRECT [9] as a two-phase framework attempts to detect false data by first leveraging a time-series of location data to DETECT suspicious data points while the CORRECT phase marks those points as missing for reconstruction algorithms. This method is not a general algorithm as it relies on additional contextual information of the particular MSC setup, whereas FSD is agnostic to the type of targeted domain. Methods surrounding matrix separation have also been successfully used for separating false data. Light Weight Low Rank and Sparse Matrix Separation (LightLRFMS) [7] is a matrix separation technique but it was only validated on a specific MCS context for environmental monitoring that does not fully consider issues pertaining to concept-drift, which FSD strives to address based on the use of LSTMs. Furthermore, various deep learning techniques have previously been applied in MSC including LSTM models towards predicting traffic flow [10] or user mobility [11]. However, these do not consider the presence of adversaries.

Banerjee et. al [5] showed that most common unsupervised learning algorithms are prone to adversarial infection and, hence, there is a need to leverage a mix of advanced machine learning models for overcoming this challenge. Our paper extends the problem disposition and investigation in [5] by trying to address issues that may be advantageous by the application of deep learning. We will apply some of the tools discussed in [5] to simulate adversaries targeting the datasets to evaluate our proposed model. Overall, there is a consensus that issues of accuracy, as a result of factors such as concept-drift and false data generation, must be addressed by leveraging deep learning based techniques as the ones employed by FSD.

III. Towards Trustworthy MCS Tasks

In MCS platforms, users can participate in the sensing process and upload their contributions to the central server, and collected can be processed by local analytic algorithms towards producing consumable data for requesting applications [12]. In this context, for a specific time window with n time steps and m sensors, we consider a dataset D containing a sequence (S) for each sensor j where $S_j = [v_{1,j}, v_{2,j}, \cdots, v_{i,j}, \cdots, v_{n,j}]$.

Threat Model: The aim of adversarial agents is to mislead the MCS applications towards considering malicious measurement values as legitimate in their services. To this end, an adversary may change the input value $v_{i,j}$ in S_j to $v'_{i,j}$, where $v'_{i,j} \neq v_{i,j}$ to maximize the distortion "$\max\{|v_{i,j} - v^{'}_{i,j}|\}$", where the distortion should be lower than a maximum allowed considered by the adversarial agent.

There are two primary adversarial attack models [13]: 1) pre-training (poisoning) attacks, and 2) post-training (evasion) attacks. In pre-training attacks, adversaries try to inject malicious data in an attempt to poison the training dataset and, thus, decrease the classification accuracy of the classifier. In the post-training attack strategy, adversaries aim at misleading trained classifiers to mis-classify samples towards a malevolent intent. Let us assume $f(x_i) = y_i$ as the mapping function to calculate/map x_i to y_i. In principle, a machine learning technique tries to minimize $|f(x'_i) - y'_i|$ which means minimizing False Positives and False Negatives. On the contrary, an adversarial attacker attempts to maximize the impact of the attack by maximizing $|f(x'_i) - y'_i|$. In the rest of the paper we refer to adversarial data as positive class of data, and legitimate data as negative class.

IV. FSD Conceptual Architecture

The main motivation behind FSD is to benefit from the relationships between samples of different time orders so as to accurately predict the values of the next time step. It uses the distance metrics for comparing the predicted data and real data, in each time step, in order to prepare samples for the one-class classification approach; i.e, draw a boundary around the allowed deviations between predicted and real data values.

Our approach consists of two main phases, namely the *Training* and *Testing* phases. In the *Training Phase*, we build a predictor for each sensory data as well as a final one-class classifier whereas during the *Testing Phase*, we feed a combination of malicious and benign testing samples in order to evaluate how good our trained models behave in separating malicious data from benign values.

A. Training Phase

Figure 1 depicts the underpinnings of the FSD training phase comprising of three steps: 1) training of LSTMs as sequential predictors for each sensor, 2) measuring deviations between predicted and real data vectors, for each time step, and 3) training an one-class classifier on vectors of the observed deviations in order to detect anomalies according to real and expected predicted values (again for each time step).

1) Training LSTMs: In this step, FSD trains separate LSTM models, M, for predicting values of each sensor, for each time step. As LSTM is useful for processing, classifying, and making predictions based on time series, we employ LSTMs on time series of sensor data in this paper. It is worth noting that the LSTMs in Figure 1 can be replaced by any time series based technique to predict next step data. This step results in m trained models based on m sensors, and M_j would be a trained model on S_j for predicting the $p_{i,j}$ value of sensor j at time step i.

2) Measuring deviations between predicted and real values: After training our LSTM models (Ms), FSD feeds all Ss in parallel (S_j to M_j), and store all predicted values ($p_{i,j}$) in

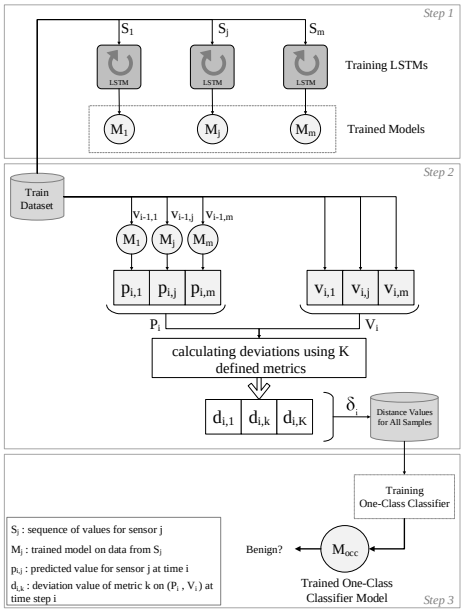


Figure 1: Training phase of FSD

matrix P where each row contains all predicted values by M_j on S_j for each time step. P_j, is column j of P showing the predicted values for sensor j for time step 1 to n, and $P_i = [p_{i,1}, \cdots, p_{i,m}]$ is a vector of all predicted values for m sensors at time step i referring to rows of P.

At time step i, we calculate deviations between predicted vector (P_i) and real vector (V_i) into a new vector $\delta_i = [d_{i,1}, d_{i,2}, \cdots, d_{i,K}]$ using K distance/similarity metrics such as Euclidean distance, cosine similarity, Manhattan distance, or other customized metrics with the ability of calculating one single scalar to show the deviation between the values of the two input vectors. Calculating the differences of predicted and real values we would have a new dataset Δ as a set of δ vectors with size K where $\delta_i = [d_{i,1}, ..., d_{i,k}, ..., d_{i,K}]$, and $d_{i,k}$ is the deviation between vector V_i and P_i calculated by distance metric k (e.g. Euclidean distance).

3) Training one-class classifier: After feeding all legitimate data to our trained models (Ms) and calculating δ_i for each time step i, we have $\Delta = [\delta_1, ..., \delta_i, ..., \delta_n]$ containing all *allowed deviations* between the predictions and the real data for all time step. Dataset Δ is suitable for one-class classifiers trying to find a boundary around the training samples in order to separate them from the data from other distributions, where M_{occ} in Figure 1 would be the trained one-class classifier.

B. Testing phase

Figure 2 shows how FSD works in real time to detect adversarial samples. For each testing time step i, FSD predicts P_i using trained models (Ms), and calculates δ_i as the distance vector between V_i and P_i. Then, M_{occ} classifies δ_i as *Benign*

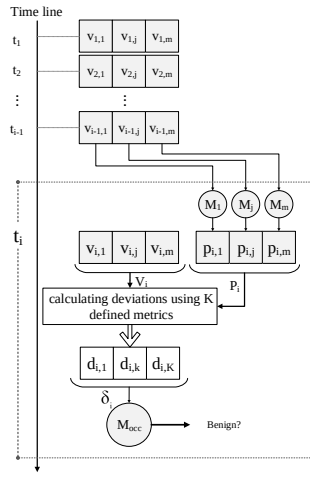


Figure 2: Testing phase of FSD

or *Malicious*. As M_{occ} is a trained one-class classifier on legitimate deviations between real and predicted values, the output dictates whether the vector of calculated deviations is legitimate.

V. Experimental Setup

In this section, we proceed with presenting our setup for implementing and evaluating FSD under various attack strategies. The dataset considered in this paper originates from *real-world measurements* collected from Data Sensing Lab [14] by sensors deployed at the Strata Clara convention center in 2013 [1]. The dataset contains sensor data from 5 different sensors namely, Temperature (Temp), Humidity (Hum), Pir, Motion and Microphone (Mic). Table I shows the distributions of each sensor type.

	Temp	Hum	Pir	Motion	Mic
μ	22.5150	36.1389	0.1514	-0.000386	5.2275
σ	1.6171	6.0058	0.3583	0.3544	5.1334

Table I: Mean (μ) and standard deviation (σ) of base dataset

LSTM and One-class Classifier Architectures

As we are dealing with sequential sensor data with specific time steps, Recurrent Neural Networks based techniques - like the leveraged LSTMs - provide an attractive root of trust. In this context, in order to better justify the reasoning behind moving from Multi Layer Perceptron (MLPs) towards more complex time oriented classification techniques, we compare the best Root Mean Square Errors (RMSE) achieved by MLP and LSTMs. Table II shows the RMSE values between predicted and real corresponding values for each sensor. We name LSTMs with 20 units as $LSTM_{best}$ as it showed the best performance in our experiments with different number of units. LSTM with one LSTM unit ($LSTM_1$) and conventional Neural Networks with 10 hidden layers (MLP_{best}) are reported to compare with $LSTM_{best}$.

	Temp	Hum	Pir	Motion	Mic
MLP_{best}	0.208	0.350	0.797	1.152	2.322
$LSTM_1$	0.037	0.081	0.347	0.521	1.832
$LSTM_{best}$	0.029	0.080	0.301	0.216	1.639

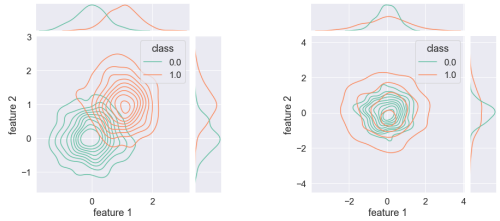
Table II: Comparing RMSEs achieved by the best trained MLP model, one-layer LSTM, and best trained LSTMs.

Adversarial Behaviour: As FSD is a framework for characterizing between inlying and outlying sensory data reports in the presence of adversarial malicious users, we assume that colluding adversaries are attacking the data collection process (poisoning attack) or the classification process (evasion attack). Therefore, we have effectively categorized attack strategies into pre-training and post-training attack strategies respectively. We measure the performance of the FSD framework based on different levels of distortion that colluding adversaries try to impose on the data trustworthiness. We consider two main attack approaches for each attack strategy: *1) distribution attacks (Attack Cases I & II)*, which manipulate mean or standard deviation to generate adversarial samples which are different from the benign ones; *2) position attacks (Attack Cases III, IV & V)*, which manipulate the position of injecting adversarial samples in the sequence of values which in turn targets the order of samples. Position attacks can only be applied after a distribution attack for changing the order of samples to be injected in the sequence.

Attack Case I: Adversaries may affect the system uncertainty about the true value of the sensory data by setting $\sigma' = \sigma$ and $\mu' \neq \mu$ which represents a malicious standard deviation equal to the standard deviation of the legitimate data points with smaller/larger μ as Equation (1) where λ is a scalar value as the deviation factor. Adversarial samples generated by $\lambda > 2$ are not attractive in our data as they are easier to detect due to their lower overlap with benign samples.

$$\begin{cases} \mu' = \mu + (\lambda \times \sigma) & 0 < \lambda \leq 2 \\ \sigma' = \sigma \end{cases} \tag{1}$$

Figure 3a compares the distributions of legitimate samples and adversarial samples generated by Attack Case I for a simple dataset with only two features, where $\lambda = 1.0$.



(a) (b)

Figure 3: Distributions of legitimate samples (class 0) vs. adversarial samples (class 1) for two features with $\mu = 0$ and $\sigma = 1.0$; (a) Attack Case I, and (b) Attack Case II.

Attack Case II: Adversaries may affect the system uncertainty by selecting an adversarial distribution based on equation 2 where the adversary's goal is to keep the same mean but changing the standard deviation.

$$\begin{cases} \mu' = \mu \\ \sigma' = \sigma + (\lambda \times \sigma) & 0 < \lambda \leq 2 \end{cases} \tag{2}$$

Figure 3b shows the distributions of a dataset with legitimate and adversarial data from Attack Case II with $\lambda = 1.0$. This attack case is designed to better reflect the real-world case scenarios where adversaries attempt to gradually change the classification behaviour by performing concept drifting by modifying the standard deviation over time.

Attack Case III: Adversaries may affect the system uncertainty about the true values by selecting a different distribution by changing the *order* of legitimate and malicious data points in the sequence of data. In this case all legitimate data comes before malicious data points in the sequence.

Attack Case IV: In this case all malicious data come before benign data points in the sequence as opposed to Case III.

Attack Case V: Adversaries may affect the system uncertainty by putting malicious batches of data in between legitimate data batches.

VI. Results and Analysis

In this section, we evaluate the performance of FSD in the presence of strong adversaries under the two aforementioned attack strategies (pre-training and post-training). We use F-Measure as our evaluation metric since it directly considers True/False Positive Rates as well as Recall and Precision metrics. The dataset is partitioned into two sub-sets of training and testing sets with 60% and 40% of data respectively.

A. Strategy 1: Impact of Adversaries in Pre-training

In the pre-training strategy, detecting malicious samples would be more difficult than in the post-training case since the classifier has been trained to recognize the adversarial samples as legitimate. Therefore, increasing the rate of adversaries has a higher impact on the classifier as it may consider all or parts of the injected malicious samples as "noise", thus, changing the overall perception of the classifier for the sensed phenomenon. Usually, the attackers tend to choose higher λ values during the pre-training attack strategy as their end-goal would be to mislead the classifier to cover a broader area and misclassify the attacking samples as legitimate in testing time. Due to limited space, we only reported the results based on changing deviations in μ' for Attack Cases III, IV and V.

1) Distribution Attacks (Cases I & II): Tables III & IV report the results for Attack Cases I & II, respectively. Each row of the tables show F-Measures for a specific λ value where the attacker generates malicious data using the previously described equations. In the pre-training strategy, increasing λ should result in a decrease in the adversarial data rate, thus, leading to a less accurate classification due to the adversarial samples affecting the classifier in order to cover malicious data as legitimate by expanding the area under the legitimate cluster (see Figure 3a). Tables III & IV clearly show this performance degradation. Similarly, increasing the rate of adversarial samples in pre-training prevents the classifier

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.813	0.768	0.727	0.718	0.704	0.668	0.652	0.634	0.613
1	0.768	0.737	0.696	0.676	0.679	0.643	0.608	0.591	0.558
2	0.721	0.701	0.662	0.635	0.613	0.601	0.579	0.556	0.506

Table III: F-measures for Attack Case I in pre-training strategy

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.755	0.741	0.719	0.700	0.681	0.661	0.641	0.597	0.564
1	0.716	0.690	0.681	0.668	0.648	0.632	0.598	0.548	0.514
2	0.678	0.668	0.665	0.658	0.621	0.607	0.565	0.516	0.468

Table IV: F-measures for Attack Case II in pre-training.

from ignoring the impact of malicious data in the training procedure. Thus more adversarial samples mean more power to mislead the trained classifier. Therefore, F-Measure of ML techniques should be decreasing when increasing the rate of adversarial data in pre-training due to the increase in quantity of False Positives and False Negatives. As expected, FSD faced performance degradation by increasing the rate of malicious samples (Adversarial Rate in Tables III & IV). However, the results are significant in comparison to conventional machine learning techniques where FSD achieved F-measures of 0.556 for Attack Case I and 0.516 for Attack Case II with a lot of malicious samples (40%) and a significant deviation ($\lambda = 2$) which is higher than the random guessing.

The results also show that FSD is more robust against the changes in mean (Attack Case II) in comparison to changes in standard deviation (Attack Case I). We expected to see higher F-Measures for $\lambda = 0.5$ as we had the same mean value with a small change in standard deviation for adversarial samples, however, we realized that the spikes in values of *pir* and *mic* over time made it difficult for FSD to assess their values in the context of a normal distribution.

2) Position Attacks (Cases III, IV & V): In Attack Case III all legitimate data appear before the malicious data in the pre-training strategy. Depending on the rate of malicious samples, the model may ignore parts of adversarial samples. As an example with 5% of adversarial data, the LSTMs model learns from 95% of legitimate data before learning from the rest of 5% malicious data, which have less impact on the final performance in comparison with 55% legitimate and 45% malicious samples. Tables V, VI and VII support the fact that the higher rates of adversarial data cause the higher negative impact on the knowledge gained by the model from the legitimate data. The results also show that for adversarial rates $\geq 45\%$, random guessing works better because FSD also learns from malicious data in training time which entails higher false negatives rate at testing time. Comparing Table V and Table VI reveals that the samples at the beginning of the sequences has more impact on the learning process. For example, 5% of malicious samples had more negative impact on F-Measure value when FSD learned from malicious samples before benign samples in comparison with Attack Case III. Attack Case V let the adversary have malicious samples in each malicious data batch, therefore more concept drift in comparison to the other cases. Increasing the rate of adversarial samples entails that the attacker has higher chance of preventing LSTMs to learn with sufficient legitimate data due to too many concept drifts on the input data.

The results show that FSD achieved higher F-Measures in detecting samples generated by Attack Case III in comparison to the two other Attack Cases. The results also prove that the superiority of FSD in detecting samples from Attack Case III diminishes by increasing λ or the rate of malicious samples. For example, based on Tables V & VI, the difference between F-Measures for $\lambda = 0.5$ and adversarial rate 5% is $(0.059 = 0.842 - 0.783)$, while for $\lambda = 2.0$ and adversarial rate 45% is only $(0.011 = 0.482 - 0.471)$. Due to page limits, we reported all position attacks using Attack Case I, however, FSD performance on cases using Attack Case II having similar trend. Moreover, FSD was more robust against attacks with one concept drift (Attack Cases III & IV) in comparison to attacks that try to trigger more drifts on the data preventing the model to learn enough knowledge from legitimate data. FSD also achieved high performance in cases with high rates of malicious samples implying superiority over conventional machine learning studied at [5].

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.842	0.736	0.701	0.687	0.672	0.651	0.646	0.626	0.561
1	0.753	0.717	0.675	0.659	0.635	0.626	0.625	0.601	0.516
2	0.718	0.688	0.651	0.631	0.616	0.608	0.596	0.565	0.482

Table V: F-measures for Attack Case III (benign first) with attack samples generated by Attack Case I in pre-training.

To give a comprehensive overview of FSD under various position Attack Cases, Table VIII compares F-Measures of Attack Cases III, IV and V. To calculate values of Table VIII, we subtracted and averaged all values of two comparing Attack Cases. For example, to calculate first row of Table VIII which compares F-Measures of Attack Case III to Attack Case IV, we first subtracted all values of Table VI from the paired values of Table V, and then calculated the average of all differences for each column shown in Table VIII. Negative values in the table shows that FSD achieved higher F-Measures for the second attack case.

B. Strategy 2: Impact of Adversaries in Post-training

This strategy generates samples for bypassing the classifier during testing time without accessing the training dataset. Therefore, we first trained the FSD on legitimate data to evaluate this strategy. In post-training strategy, it is easier for a

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.783	0.719	0.683	0.676	0.663	0.647	0.638	0.612	0.554
1	0.729	0.683	0.649	0.639	0.622	0.602	0.611	0.581	0.497
2	0.702	0.647	0.612	0.591	0.585	0.585	0.571	0.559	0.471

Table VI: F-measures for Attack Case IV (malicious first) with attack samples generated by Attack Case I in pre-training.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.782	0.735	0.699	0.648	0.624	0.610	0.601	0.571	0.492
1	0.758	0.683	0.646	0.623	0.593	0.574	0.567	0.534	0.443
2	0.717	0.655	0.611	0.589	0.563	0.541	0.511	0.486	0.421

Table VII: F-measures for Attack Case V with attack samples generated by Attack Case I in pre-training.

	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
CaseIII-CaseIV	0.033	0.030	0.027	0.023	0.017	0.017	0.015	0.013	0.012
Case V-Case III	-0.018	-0.022	-0.023	-0.039	-0.047	-0.053	-0.062	-0.067	-0.067
Case V- Case IV	0.014	0.008	0.004	-0.015	-0.030	-0.036	-0.047	-0.053	-0.055

Table VIII: An overview of the F-Measures of FSD under different Attack Cases III, IV and V in pre-training.

trained classifier to detect non-overlapping malicious samples as the classifier has the knowledge to cover the legitimate area. Therefore, an adversary with this strategy tends for lower λ since they have to follow the distributions of legitimate data with minor changes for a successful evasion attack.

1) Distribution Attacks (Cases I & II): Table IX shows the results of classifying poisoned testing dataset by Attack Case I. FSD achieved higher F-Measures when we increased the rate of malicious examples (Adversarial Rate) because it was trained on legitimate data, so detecting malicious samples would be easier if the adversary inject more samples. This is because malicious data may create a new cluster with new distribution, so it is easier to detect them in cases with enough adversarial samples (higher rates of adversarial samples).

Comparing Table III in pre-training to the results from post-training (Table IX) shows opposite trends for both increasing the rate of adversaries and increasing the deviation factor (λ). Opposite to pre-training attacks, increasing the rate of adversarial samples may make it easier for the classifier to separate malicious data in post-training strategy. Higher deviation factors make an adversarial cluster with higher distances to the legitimate cluster resulting to higher F-Measures for post-training and lower F-Measures in pre-training.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.551	0.582	0.598	0.606	0.629	0.648	0.677	0.698	0.743
1	0.573	0.595	0.621	0.643	0.678	0.688	0.714	0.735	0.763
2	0.592	0.600	0.642	0.666	0.697	0.701	0.745	0.765	0.815

Table IX: F-measures for Attack Case I in post-training.

The reported F-Measures in Table X indicate that FSD achieved higher performance in cases where adversaries change σ' without updating μ' (Attack Case II). Similar to Attack Case I, there is an increase in F-Measure along with the increase in λ and number of malicious data points in the post-training attack strategy. Comparing the results of Tables IX and X, we can see that FSD achieved better performance in cases where adversarial data were generated by updating σ' with the same mean. FSD achieved values of 0.551 and 0.566 for Attack Case I and Attack Case II, respectively, which are higher than the random guessing in our worst-case experiments with $\lambda = 0.5$ and 45% of adversarial data.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.566	0.586	0.610	0.638	0.689	0.743	0.766	0.796	0.841
1	0.636	0.656	0.667	0.684	0.706	0.756	0.776	0.819	0.866
2	0.680	0.685	0.699	0.716	0.740	0.770	0.785	0.829	0.906

Table X: F-measures for Attack Case II in post-training

2) Position Attacks (Case III, IV & V): Table XI describes F-Measures of detecting adversarial samples when all benign samples appeared before malicious samples in post-training. More adversarial samples with higher deviation means more distance between benign and malicious clusters. Therefore, FSD achieved high F-measure of 0.903 in *post-training* strategy when the adversaries injected a lot of malicious samples (45%) with high deviation of 2.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.599	0.615	0.651	0.681	0.715	0.734	0.781	0.800	0.832
1	0.632	0.655	0.681	0.713	0.759	0.777	0.810	0.833	0.866
2	0.678	0.681	0.702	0.734	0.774	0.795	0.832	0.851	0.903

Table XI: F-measures for Attack Case III (benign first) with attack samples generated by Attack Case I in post-training.

Table XII shows F-Measures of FSD in testing dataset under Attack Case IV in post-training. As LSTMs work based on the history of data elements in time series data, the attacker in Attack Case IV attempts to affect the initial weights of LSTMs. Therefore, we see lower F-measures in comparison to Attack Case III. Comparing Tables XI & XII shows that FSD is more efficient if the adversary positions all benign samples at the beginning of testing data.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.487	0.536	0.588	0.609	0.637	0.666	0.696	0.743	0.785
1	0.534	0.597	0.601	0.652	0.705	0.718	0.767	0.769	0.800
2	0.597	0.607	0.648	0.679	0.722	0.741	0.783	0.799	0.851

Table XII: F-measures for Attack Case IV (malicious first) with samples generated by Attack Case I in post-training.

Table XIII reflects F-Measures of FSD where the adversary positions the malicious data in between benign samples. FSD showed better performance under this attack in comparison to Attack Case IV, however the results of Attack Case V are more or less similar to Attack Case III. F-Measures of higher than 0.785 with 45% of adversaries and $\lambda = 0.5$ is significant for FSD under Attack Cases III-V in comparison to conventional machine learning at [5].

Table XIV gives an overview of F-Measures for Attack Cases III, IV and V. The first tow of the table shows that FSD achieved higher F-measures when the adversaries put benign samples before the attacking samples. The second row of Table XIV shows that increasing the rate of adversarial samples to more than 25% in Attack Case V makes the detection easier for FSD to detect malicious samples in comparison to Attack Case III with adversarial rate of $\lambda > 25\%$.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
0.5	0.579	0.611	0.658	0.683	0.729	0.749	0.780	0.803	0.860
1	0.592	0.631	0.676	0.704	0.748	0.771	0.814	0.838	0.878
2	0.606	0.661	0.680	0.732	0.771	0.790	0.839	0.859	0.901

Table XIII: F-measures for Attack Case V post-training attack strategy with attack samples generated by Attack Case I.

	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
Case III-Case IV	0.097	0.070	0.065	0.062	0.061	0.060	0.059	0.057	0.055
Case V-Case III	-0.044	-0.016	-0.006	-0.003	0.000	0.001	0.003	0.005	0.012
Case V-Case IV	0.053	0.054	0.059	0.059	0.061	0.061	0.062	0.063	0.067

Table XIV: An overview of the F-Measures of FSD under different Attack Cases III, IV and V in post-training.

VII. Discussion and Critique

As it is commonly the case for any relatively young research area, the landscape of MCS for IoT is fragmented into various families based on the emerging research challenges. Undoubtedly, data trustworthiness is a prominent challenge with unprecedented number of consequences. We consider this paper as the first step towards the development of a holistic framework, which will improve data trustworthiness by leveraging advanced deep learning capabilities.

For the evaluation, the system uses different distance measures comprising data vectors of all 5 features originating from data sources of various trustworthiness levels; containing both benign and malicious data samples. This segregation is denoted by a scalar applied to the standard deviation of each feature ensures that the data distribution considered for each feature is balanced. However, there are a number of challenges to be considered: First of all, features do not behave the same way over time as reflected in their distribution presented in our evaluation due to changing statistical properties over time. Typically, a feature can be modelled by a normal distribution, however, some features possess a lot of spikes and shifts frequently which makes them harder to predict. Furthermore, this affects the overall performance as all features are merged to one representative vector, meaning that features with varying behavior over time can affect the overall performance negatively, as was also reflected in our results.

Secondly, with colluding adversaries attacking the data collection process by feeding the classifier with falsified data, it is evident that not only does the order of how the data is provided matter significantly in the system's ability to classify malicious users, but so does the time window size. This is why the use of LSTM agents is beneficial as it considers the sequential order of data. In this work, we have investigated the impact that this ordering of data has along the observed distribution. However, in the future it may be fruitful to also explore the open challenges regarding the impact of selecting various time windows and consider the influence this might have on the performance of the classifier. Recall that if the time window is increased, we effectively increase the overall knowledge of the classification process. However, it should be noted that larger time windows might introduce sever scalability issues for real-world applications.

While the current status and functionality of FSD has been evaluated against advanced data poisoning attacks, it is also our intention to perform a detailed analysis considering also even more intelligent adversaries targeting different facets of the MCS paradigm such as evasion attacks and more advanced data poisoning techniques like the one proposed by Miao et. al [15]. As another research direction, we may study how extracted features from the sequential information between data elements from other sensors can contribute to detect malicious samples in a sensor.

VIII. Conclusion

Data trustworthiness has always been one of the main concerns that current MCS platforms are facing. FSD works based on measuring the deviation between the predicted and the real received sensor values. We used LSTMs and One-Class SVM to build FSD. However, it is simply possible to replace them with alternative techniques working on time series and one-class classification. As an intelligent framework, FSD improves data trustworthiness by providing the capability of detecting falsified data generated by adversaries under both pre-training and post-training attack strategies. In our experiments we designed five attack cases consisting of two distribution attack cases and three position attack cases to evaluate both attack strategies. FSD achieved higher performance when it visits more benign samples before malicious samples in both studied strategies.

IX. Acknowledgment

This work was supported by Innovation Fund Denmark (IFD) under SecDNS project and the European Commission under the STAR project, Grant Agreements no. 956573.

References

[1] T. Giannetsos, S. Gisdakis, and P. Papadimitratos, "Trustworthy people-centric sensing : Privacy, security and user incentives road-map," in *2014 13th Annual Mediterranean Ad Hoc Networking Workshop, MED-HOC-NET 2014 :*, 2014, pp. 39–46, qC 20150108.

[2] N. P. Owoh and M. M. Singh, "Security analysis of mobile crowd sensing applications," *Applied Computing and Informatics*, vol. ahead-of-print, no. ahead-of-print, Jul. 2020.

[3] A. S. Chivukula and W. Liu, "Adversarial learning games with deep learning models," in *Int. Joint Conference on Neural Networks*, 2017.

[4] R. C. Cavalcante, L. L. Minku, and A. L. Oliveira, "Fedd: Feature extraction for explicit concept drift detection in time series," in *Int. Joint Conference on Neural Networks*, 2016.

[5] N. Banerjee, T. Giannetsos, E. Panaousis, and C. Cheong Took, "Unsupervised learning for trustworthy iot," 07 2018, pp. 1–8.

[6] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, "Mobile crowd sensing and computing," *ACM Computing Surveys*, vol. 48, no. 1, pp. 1–31, Sep. 2015.

[7] X. Li, K. Xie, X. Wang, G. Xie, D. Xie, Z. Li, J. Wen, Z. Diao, and T. Wang, "Quick and accurate false data detection in mobile crowd sensing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1339–1352, 2020.

[8] L. Cheng, L. Kong, C. Luo, J. Niu, Y. Gu, W. He, and S. Das, "Deco: False data detection and correction framework for participatory sensing," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, Jun. 2015.

[9] B. Wang, L. Kong, L. He, F. Wu, J. Yu, and G. Chen, "I(TS, CS): Detecting faulty location data in mobile crowdsensing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Jul. 2018.

[10] Y. Tian, K. Zhang, J. Li, X. Lin, and B. Yang, "Lstm-based traffic flow prediction with missing data," *Neurocomputing*, vol. 318, 2018.

[11] W. Yang, G. Sun, X. Ding, and X. Zhang, "Budget-feasible user recruitment in mobile crowdsensing with user mobility prediction," in *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 2018, pp. 1–10.

[12] R. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[13] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, Nov. 2019.

[14] D. S. Lab, ""strata santa clara dataset"." [Online]. Available: http://datasensinglab.com/data/

[15] C. Miao, Q. Li, H. Xiao, W. Jiang, M. Huai, and L. Su, "Towards data poisoning attacks in crowd sensing systems," 06 2018, pp. 111–120.

Detecting Ambiguous Phishing Certificates using Machine Learning

Sajad Homayoun
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
sajho@dtu.dk

Kaspar Hageman
Department of Electronic Systems
Aalborg University
Aalborg, Denmark
kh@es.aau.dk

Sam Afzal-Houshmand
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
saaf@dtu.dk

Christian D. Jensen
DTU Compute
Technical University of Denmark
Kongens Lyngby, Denmark
cdje@dtu.dk

Jens M. Pedersen
Department of Electronic Systems
Aalborg University
Aalborg, Denmark
jens@es.aau.dk

***Abstract*—Recent phishing attacks have started to migrate to HTTP over TLS (HTTPS), making a phishing web page appear safe to the user's browser despite its malicious purpose. This paper benefits from both digital certificates and domains related data features to propose machine learning-based solutions to predict digital certificates involved in HTTPS as phishing or benign certificates. In contrast to previous works that consider this a binary classification problem, we take into account that a certificate can be partially benign and phishy simultaneously. We propose a multi-class classifier and a regressor to classify these ambiguous certificates, in addition to benign and phishing certificates, where the 'phishyness' of a certificate is expressed as a value between 0 and 1 for the regressor. We apply our method to a set of certificates obtained from certificate transparency logs and show that we can classify them with high performance. We extend our validation by evaluating the performance of the model over time, showing that our model generalizes over time on our training data set.**

***Index Terms*—Digital Certificate, Phishing, Machine Learning, Feature Extraction**

I. Introduction

Phishing is still one of the most common attacks which has a plethora of reported cases on a daily basis [1]. In these attacks, victims are persuaded in disclosing sensitive information by an attacker that impersonates a legitimate organization or person. Modern and more advanced attackers have begun to use phishing websites offering legitimate digital certificates as a way to persuade the users into diverging their information willingly, by making the websites look more legitimate through visible padlock icons in browsers. Although phishing websites with legitimate certificates can be more effective than their unencrypted counterparts, they have the drawback from the attacker's perspective that they have are visible in publicly available certificate transparency (CT) logs.The inclusion of certificates in these logs is a requirement to be a trusted website in most modern mainstream browsers [2]. Monitoring CT logs for newly-issued certificates may enable the detection of malicious phishing websites using legitimate certificates in the earlier stages of the attack. In fact, prior research has proposed automated solutions for detecting digital certificates being involved in phishing attacks [3]–[6]. Most of previous research has tackled this as a binary classification problem, with certificates being considered either *malicious* or *benign*. Identifying a high-quality ground truth is challenging, because the label of a certificate tends to based on the label of the domain name or website that serves a given certificate. Not only is the nature of the domain not always clear-cut benign or malicious [3], [7], but there may also be conflicting domain labels for a given certificate [8].

In this paper, we propose a novel machine learning technique to address the issues with this task in the state of the art. In contrast to prior work, our solution considers the classification of certificates to be a multi-class classification problem, and is capable of finding certificates that fit neither the traditional benign or malicious labels, but are rather considered ambiguous or 'conflicted'. More specifically, the contributions in this paper are summarized as follows:

- we present a novel labeling mechanism that takes into account the individual labels of the domains that are being covered by a certificate,
- our study is not limited to benign and phishing certificates but also considers certificates covering both benign and phishing domains,
- We combine all features described in prior works to build a pool of features for machine learning tasks,
- we study how our features can predict a *phishyness score* for each certificates,
- we validate our methods using a time-based cross validation scheme and show that our classifier and regression models achieve a high performance.

This paper is divided as follows. Section II gives brief background on digital certificates and Section III reviews some related works. Section IV describes our data collection

pipeline extracting certificate/domain related features. Section V & VI presents our classification and regression scenarios, respectively. Section VII evaluates our scenarios using a time-based approach. We discuss the achievements and limitations in Section VIII, and conclude the paper in Section IX.

II. Background

The Transport Layer Security (TLS) protocol suite provides encryption functionality that other networking protocols such as HTTP and IMAP. Besides encrypting the communication between a client and server, TLS guarantees authenticity by operating under a public key infrastructure (PKI). When establishing a connection, a server provides a digital certificate (as defined in the X.509 format), which consists of a public key (whose private key is only known to the server operator), a set of identities (usually one or more domain names) and a signature from a trusted third-party. These third parties, or Certificate Authorities (CA), only sign new certificates after successfully validating that the requestor of the certificate can demonstrate ownership of all identities to be included in the certificate. Clients can verify the authenticity of a certificate by validating that the signature was created using a private key that the client inherently trusts, as part of the certificate root store installed on their system. Originally, a certificate contained a single *subject* field, which indicated for what domain name the certificate was valid. An extension was later introduced to support multiple domain names to be included, named the `Subject Alternative Name` extension, and the domain names embedded in a certificate are therefore commonly refered to as SANs.

In 2011, two CAs issued certificates for high-profile domain names for malicious actors, which allowed these actors to perform large-scale impersonation attacks. As a response, the Certificate Transparency (CT) framework was developed with the intention of monitoring the certificate issuance behavior of the CAs. In this framework, CAs are encouraged to submit newly-issued certificates to CT logs, publicly-available repositories of certificates. Browser vendors started to require certificates to be included in these logs, and as a results the CT logs capture nearly all certificates that are issued worldwide.

III. Related Work

Artificial intelligence based approaches for classifying malicious certificates have been studied during the past few years. Inspired by classifiers for URL detection, there have been works on classifiers that utilize the information that can be gained from the certificates itself which is encompassed in the work presented in this paper. Mohammad *et al.* provide a dataset that illustrates how using the certificate issuer from HTTPS information can be used to high avail which has bled over to multitudes of other works [9]. It ought to be noted that certificate information often needs additional information extraction of viable features due to the limited information offering as compared to URL databases [10].

Across works in the area of detecting phishing websites based on certificates, there is a recurring theme of feature engineering. Mishari *et al.* proposed a selection of certificate features from phishing websites to be used for training a random forest, decision tree and nearest neighbor to detect phishing website which denoted an accuracy above 85% [11]. A more holistic real-time version with a similar approach and results was proposed by Dong *et al.* where the framework of feature extractor, classifier and decision process was included with phishing information used to train the same models in addition to naive bayes tree, logistic regression, decision table and k-nearest neighbor [12]. A deep neural network version was also proposed for classification by Dong *et al.* which denoted an accuracy above 95% [13]. Relying on deep learning, Torroledo *et al.* use a long short-term memory (LSTM) based model for detection [6]. Recently, Drichel *et al.* propose a pipeline where they could easily test a lot of these classifiers using CT log data which may help in classifier selection process [3].

The security research community tends to rely on information from domain names to provide a label for certificates that cover the given domain names. Typically, a list of popular domains serves as a ground truth for benign domains, and lists of phishing domains or URLs (e.g., PhishTank [14]) as benign domains. Certificates that are served by these two respective domain group can as a result be labeled as benign and phishing as well. Hageman *et al.* showed that labeling certificates in such as fashion may result in mislabeling [8]. Content Delivery Networks (CDNs) such as CloudFlare and Incapsula that provide HTTPs based protection services issue certificates for sets of domains originating from different owners. In case this set of domains include both malicious and benign domains, labeling these certificates is a challenge.

Even though a significant effort was made by the security community towards the detection of certificates involved in malicious activity, to the best of our knowledge no one has recognized it as an ambiguous problem that should not be tackled as a binary classification problem.

IV. Data Collection

To train and validate our approach, we rely on a vast number of labeled certificates. First, we describe how we extract a relevant label and feature space from a certificate, and then explain how we obtained a large dataset of certificates.

A. Feature extraction and labeling

Figure 1 shows the feature extraction and label extraction process. For each unlabeled certificate, and a set of labeled domains, it produces a feature space and a label for the certificates. This set of labeled domains is prepared in advance and contains both benign (i.e., domains that are – with high confidence – have not been part of a phishing attack) and phishing domains (i.e., domains that have been observed as part of a phishing attack). The resulting feature space is a combination of features extracted from the certificate itself, and aggregated features extracted from the list of SANs that are covered by the certificate. For the feature extraction components, we rely on a combination of features that have

been used in prior research [3]–[5] or are derived from insights from other work [8] resulting in 107 features. Due to page limit, we have uploaded a list of our features at our page on the internet[1]. The first 58 features are extracted from the X.509 certificates themselves, and include features such as the signing parameters (e.g., key size, signature algorithm), extensions (i.e., the presence and content of certain X.509 extensions) and the composition of the subject field. The remaining 49 features are extracted from the lexical properties of SANs covered by the certificate, such as the presence of particular keywords or features related to the characters composition and diversity in the string. Each certificate covers a variable number of SANs, and the feature space of these individual SANs are condensed in a fixed-length feature space. We rely on simple statistical functions (i.e., min, max, mean, median) to summarize, and express the diversity of, the numerical domain features and compute a ratio for condensing binary domain features. The dataset contains a significant number of duplicate samples, which we filter out. It is not uncommon for certificates to be renewed after they expire, covering the same SANs and being signed by the same CA with the same parameters, resulting in all our extracted features to remain identical[2].

The label of a certificate is inferred from the collection of the labels from the SANs. Depending on which model is being trained, the label is one of three classes (benign, phishy or conflicted) or a continuous "phishyness" score. In the first case, a certificate is benign or phishy when the list of SANs is only composed of benign or phishing domains respectively (and may include unlabeled domains as well). A certificate covering both at least one benign and phishy domain is considered conflicted, as it is not trivial to claim the maliciousness of the certificate. In the latter case, we use a function of all SANs covered by the certificate for computing a phishyness core. This score (s_i) is expressed as a ratio of the number of phishing domains (p_i) and phishing domains, benign domains (b_i) and unlabeled domains (u_i):

$$s_i = \frac{p_i}{p_i + b_i + u_i} \tag{1}$$

Note that the label extraction is only done during the training process, and not during the operations of the framework.

B. Dataset

To train and validate our machine learning models, we collected a vast collection of certificates. This requires a set of certificate for which a ground-truth is known. We can rely on known phishing and benign domains and infer the ground-truth of certificates issued for those domains. Under the assumption that a highly-popular domain is inherently abused for phishing attacks (e.g., `youtube.com`), we rely on the top one million most popular domains from the Tranco list [15] as a ground truth of benign domains. For establishing a ground truth of phishing entities, we collect phishing URLs from the eCrime Exchange (ECX) platform [16], a platform in which various anti-phishing organizations share newly identified phishing URLs with one another. The root domains from those URLs (e.g., `example.org` from `https://www.example.co.uk?help`) form the basis of our set of phishing domains. There is an overlap between this set of domains and the benign domains, since some popular domains host some user-generated phishing content, such as Google Forms and Facebook. As such, we remove any of the top 1 million domains from the ECX domains to form our ground truth of phishing domains. Furthermore, we take into account that the nature of a phishing domain can change over time (e.g., a domain may have been registered for benign purposes for years, after which it was registered by phisher and used for hosting phishing content). It is common for phishing domains to only be abused for several days [17]. In the label aggregation phase of Figure 1, a domain is only considered "phishy" in the context of a particular certificate, if the validity period overlaps with the identification of a phishing URL associated with the domain in the ECX platform.

Table I: Collected dataset after removing duplicates.

Samples Type	Number of Samples
Benign Certificates	213,353
Phishing Certificates	46,256
Conflicted Certificates	15,578

Similar to [3], we rely on certificates from the CT logs as a basis for our ground truth. From both of our domain sets, we sampled 10,000 domains each, and collected all certificates that cover any of these 20,000 resulting domain names [8]. As a result, our dataset contains not only the certificates that are currently deployed on web servers – a common method for related work to retrieve their certificate data from [5] –, but also historical data and certificates used for non-HTTPS related services, such as mail servers. The certificates were collected from the Censys search engine [18], which provides extra information to these certificates, most notably the validation status of three root stores (Microsoft, Mozilla's NSS and Apple). We discard all certificates that, according to Censys, do not have any valid certificate chain to any of the three root stores.

V. Scenario 1: Distinguishing Between Phishing and Benign Certificates

Figure 2 depicts the underpinnings of a multi-class classification training and testing phase. The training phase comprises two steps: (1) feature engineering and preprocessing and (2) training the multi-class classifier. The former receives benign (B), phishing (P), and conflicted (C) certificates and prepares data format for the training algorithm and filters features with low variance or constant values. This step is also responsible for normalizing the values of different features. The latter outputs a trained classifier to separate between phishing, benign, and conflicted certificates. The training step is not limited to

[1]https://phish-certs.github.io/

[2]The only distinction between these certificates are the timestamps when they become active and expire.

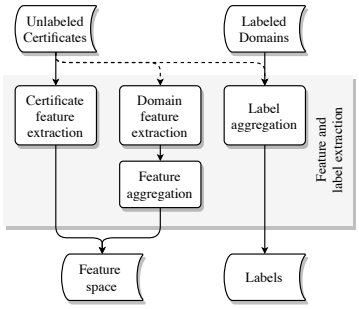


Figure 1: The feature engineering and preprocessing pipeline that turns unlabeled certificates and labeled domains in a labeled feature space. The dashed lines between the certificate dataset and the processing modules represent the set of SANs covered by the certificate.

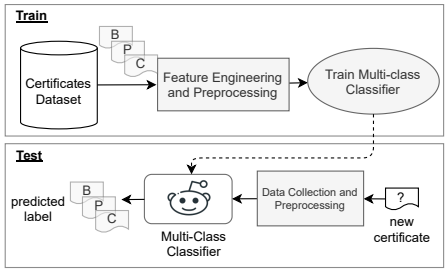


Figure 2: Multi-class classification to distinguish between Phishing, Benign and Conflicted certificates.

certain multi-class classification algorithms. In the test phase, for classifying new certificates, we first have a data collection and preprocessing step to extract certificate related and domain related features for feeding the multi-class classifier. Then the classifier outputs a label for the new certificate.

A. Experimental Results

To show the performance of our extracted features in separating phishing, benign and conflicted certificates, we selected five well-known classification algorithms namely *stochastic gradient descent (SGD)*, *k-nearest neighbors (kNN)*, *random forest (RF)*, *decision tree (DT)* and *support vector machines (SVM)*. We used *VarianceThreshold* from *sklearn* [19] for features selection with $threshold = (.9 \times (1 - .9))$. We used simple classification algorithms to show the robustness of our algorithms in classifying phishing certificates. To avoid the complexity of parameter tuning for each algorithm, we used default parameters given by *sklearn* Python library. As kNN needs k as a required parameter, we set k as the square root of the number of training samples ($k = \sqrt{N}$). We use F1 Score as our comparison metric because it expresses the precision and recall in a single metric [20]. Table II compares our trained classifiers with different metrics for 5-fold cross

Table II: F1 Score of the classifiers under 5-fold evaluation.

Fold	SGD	KNN	DT	RF	SVM
1	0.78	0.74	0.96	0.99	0.84
2	0.76	0.74	0.97	0.99	0.84
3	0.75	0.74	0.97	0.99	0.83
4	0.79	0.74	0.97	0.98	0.84
5	0.76	0.74	0.97	0.99	0.84
avg.	0.77	0.74	0.97	**0.99**	0.84

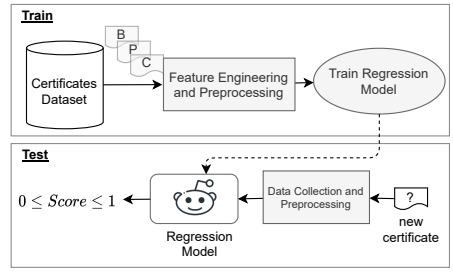


Figure 3: Regression model to predict phishyness score between 0 and 1 for each certificate.

validation. Our evaluation shows that our proposed features for detecting phishing certificates can help to build high quality classifiers. The RF classifiers generated higher performance in comparison to other classifiers.

VI. Scenario 2: Predicting Phishyness Scores

Figure 3 explains training and testing phases of a regression model to predict a phishyness score for each certificate. Similar to *Scenario 1*, this scenario has one step for preprocessing and one step for training the regression model. In the preprocessing step we calculate a phishyness score for each certificate based on equation (1). This step is also responsible for normalizing the values of different features as a data preparation task. The output would be a phishyness score for each certificate which can be between 0 and 1. In the test phase, for predicting a phishyness score for new certificates, the data collection and preprocessing step extracts all required features (certificate related and domain related features), and then the trained regression model outputs a score.

A. Experimental Results

To show the performance of our extracted features, we applied different regression algorithms on our dataset. To avoid parameter tuning of different algorithms, we applied each algorithm using default parameter set from *sklearn* [19]. Table III compare the results achieved by different algorithms such as *lasso regression*, *ridge regression*, *ElasticNet*, *random forest regressor (RFR)*, *decision tree regressor (DTR)* and *bagging regressor (BR)*. We use *Root Mean Square Error (RMSE)* given by $\sqrt[2]{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$ as it is commonly used in regression analysis to verify experimental results. RMSE is the standard deviation of the residuals, which is errors between real value

Table III: RMSE of regression models under 5-fold evaluation.

Fold	Lasso	Ridge	ElasticNet	RFR	DTR	BR
1	0.37	0.2	0.37	0.04	0.06	0.05
2	0.37	0.2	0.37	0.04	0.06	0.05
3	0.37	0.2	0.37	0.04	0.06	0.04
4	0.37	0.2	0.37	0.04	0.06	0.04
5	0.37	0.2	0.37	0.04	0.06	0.05
avg.	0.37	0.20	0.37	**0.04**	0.06	0.05

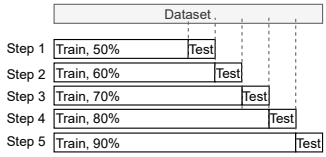


Figure 4: Time-based cross validation with 10% test data.

(y) and predicted value ($\hat{y}$) for all samples (n). *RFR* achieved the best RMSE equal to 0.04.

VII. Validation

The CT framework was created in 2011 and as a result, the CT logs consist of certificates spanning almost a decade. In cross validation, both the training and validation sets included mixed samples from different periods in time. As such, this validation does not evaluate how well the model generalizes to changes in the TLS ecosystem over time. An example of a major shift was the introduction of *Let's Encrypt*, the first certificate authority that issued certificates for free fully automated, which suddenly enabled small websites to serve their content over HTTPS.

We perform a time-based cross validation to evaluate the generalization of our models over time. In this evaluation, we take the first 50% of our certificates, as defined by their validity date, and produce the performance metrics over the next 10% of certificates. We repeat this process by taking the first 60, 70, 80 and 90% of certificates and test on the next 10% (see Figure 4). Tables IV show the F1 scores of time-based validation for the classifiers. Comparing Tables II & IV proves that our classifiers should be retrained on need data as there is a decrease in the F1 Score for classifying recent certificates. Our trained random forest classifier could achieve an F1 score of 0.84, while it could get to 0.99 in our 5-fold cross validation (II). The kNN model gave the worst results in both 5-fold and time-based cross validations, which shows that our feature space and dataset need more complex classification to separate phishing, benign and conflicted samples.

Table V describes RMSE of each studied regression model. Our time-based evaluation shows that the RFR and BR as ensemble-based algorithms have achieved RMSE of 0.05, which is best among our regression algorithms. The lasso and ElasticNet regression algorithms achieved the worst RMSE, which is 0.33. We are interested in a more in-depth look into the errors that RFR as our best model produces. As such, we calculate the error level (i.e., the absolute difference between

Table IV: F1 Score the classifiers under time-based evaluation.

Step	SGD	KNN	DT	RF	SVM
1	0.81	0.60	0.79	0.85	0.66
2	0.75	0.59	0.84	0.85	0.62
3	0.72	0.59	0.77	0.84	0.65
4	0.72	0.58	0.79	0.81	0.61
5	0.74	0.59	0.81	0.83	0.65
avg.	0.75	0.59	0.80	**0.84**	0.63

Table V: RMSE of the regression models under time-based evaluation.

Step	Lasso	Ridge	ElasticNet	RFR	DTR	BR
1	0.37	0.21	0.37	0.05	0.07	0.05
2	0.34	0.18	0.34	0.04	0.06	0.05
3	0.31	0.17	0.31	0.04	0.06	0.04
4	0.34	0.20	0.34	0.05	0.08	0.06
5	0.30	0.20	0.30	0.05	0.08	0.06
avg.	0.33	0.19	0.33	**0.05**	0.07	**0.05**

the predicted value and the real value) for each sample. We plot these figure in a cumulative distribution function in Figure 5. The figure illustrates for instance that 93.5% of samples has an error level of smaller than 0.01, which we believe gives an acceptable approximation of the actual value.

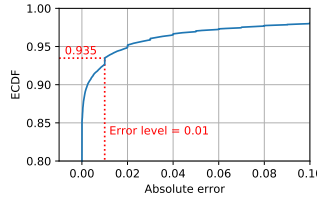


Figure 5: ECDF of the error level of real vs. predicted errors for all test samples for time-based cross validation for all test samples of all steps at Figure 4 (n=123,835).

VIII. Discussion

Our results have shown that our proposed approach can classify with a high performance, and generalizes well over time on historical data. We deliberately selected more classical and simpler classifier and regression models over more novel models, such as deep neural networks, to show that the selected features are powerful and avoid parameter settings.

a) Adverserial robustness: A major challenge for employing machine learning models is an adversarial environment, and we should consider the robustness of the model against behavioral changes of phishers trying to evade our model. Evasion here is the ability of an attacker to modify the feature space of a certificate to circumvent a 'phishing' label to be produced by the classifier. A number of features are domain name independent, and are controlled by the CA rather than the phisher (who merely requests the issuance of the certificate), and can therefore only be influenced by the

attacker by requesting their certificate from a different CA, which may have monetary consequences. The domain-related features in the feature space are derived from *all* SANs covered by the certificate. These features are changed by a phisher by requesting certificates for different sets or, or even individual, domain names, which has once again a monetary impact and may also impact the hosting infrastructure that phishers resort to. As future work, we consider evaluating the performance of our proposed approach in the absence of domain-related features or CA related features to emulate eliminating features due to the evasion strategies of phishers.

b) Ground truth challenges: As described in Section III, our work is not the first attempt at certificate classification for various security purposes. However, to the best of our knowledge we are the first to acknowledge that there can be conflicts in the label for a certificate. As a result, it is difficult to compare our results with prior work in a fair representative manner.

By monitoring CT logs, we are merely observing snapshots of a domain, and miss the changes of the domain afterwards. Phishers are known to compromise (i.e., hack) existing benign websites, re-purposing them for malicious purposes. As such, we may be labeling certificates as phishing or conflicted due to a domains being reported as phishing domains months after the certificate was issued, even though the certificates at time of issuance should have been labeled differently. We devised a method to express the maliciousness of a certificate based on the composition of labels of the domains covered by the certificate. Even though this method provides a continuous scale to put certificates on, one should be careful with relying on the results for any automated decision making. Blocking traffic to web servers serving these certificates may block benign traffic and disproportionately hit organisations that provide security services. We believe that the results can instead be highly valuable as a warning signal for security researchers or regulators to follow up on manually.

IX. Conclusion

In this paper, we proposed a novel method to automatically classify digital certificates as benign, as used in phishing attacks, or assign it an conflicted label. Our work is motivated by prior work on phishing certificate classification and on the existence of ambiguous certificates that are associated with both benign and phishing domains. We consider both a (1) multi-class classification problem, where certificates can be benign, phishy or conflicted, and (2) a regression problem where certificates have a phishyness score. By training different machine learning models in both scenarios, we show a highly performant system. Furthermore, we evaluate the resulting classifiers and regressors using time-based cross validation to show that our approach generalizes decently over time.

X. Acknowledgment

This work was supported by Innovation Fund Denmark (IFD) under the SecDNS project. The authors acknowledge researchers at CSIS Security Group A/S (https://csis.com/) for their constructive recommendations on this project.

References

[1] FBI Internet Crime Complaint Center, "Internet crime report 2020," 2020. [Online]. Available: https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf

[2] B. Laurie, "Certificate transparency," *Commun. ACM*, vol. 57, no. 10, p. 40–46, Sep. 2014. [Online]. Available: https://doi.org/10.1145/2659897

[3] A. Drichel, V. Drury, J. von Brandt, and U. Meyer, "Finding phish in a haystack: A pipeline for phishing classification on certificate transparency logs," in *The 16th International Conference on Availability, Reliability and Security*, 2021, pp. 1–12.

[4] E. Fasllija, H. F. Enişer, and B. Prünster, "Phish-hook: Detecting phishing certificates using certificate transparency logs," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2019, pp. 320–334.

[5] J. Li, Z. Zhang, and C. Guo, "Machine learning-based malicious X.509 certificates' detection," *Applied Sciences*, vol. 11, no. 5, p. 2164, 2021.

[6] I. Torroledo, L. D. Camacho, and A. C. Bahnsen, "Hunting malicious TLS certificates with deep neural networks," in *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 64–73. [Online]. Available: https://doi.org/10.1145/3270101.3270105

[7] S. Maroofi, M. Korczynski, C. Hesselman, B. Ampeau, and A. Duda, "COMAR: Classification of compromised versus maliciously registered domains," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Sep. 2020. [Online]. Available: https://doi.org/10.1109/eurosp48549.2020.00045

[8] K. Hageman, E. Kidmose, R. R. Hansen, and J. M. Pedersen, "Can a TLS certificate be phishy?" in *18th International Conference on Security and Cryptography, SECRYPT 2021*. SCITEPRESS Digital Library, 2021, pp. 38–49.

[9] R. Mohammad, F. Thabtah, and T. Mccluskey, "Predicting phishing websites based on self-structuring neural network," *Neural Computing and Applications*, vol. 25, pp. 443–458, 08 2013.

[10] U. Meyer and V. Drury, "Certified phishing: Taking a look at public key certificates of phishing websites," pp. 211–223, Aug. 2019. [Online]. Available: https://www.usenix.org/conference/soups2019/presentation/drury

[11] M. Mishari, E. De Cristofaro, K. Eldefrawy, and G. Tsudik, "Harvesting SSL certificate data to mitigate web-fraud," *CoRR*, vol. abs/0909.3688, 01 2009.

[12] Z. Dong, A. Kapadia, J. Blythe, and L. J. Camp, "Beyond the lock icon: real-time detection of phishing websites using public key certificates," in *2015 APWG Symposium on Electronic Crime Research (eCrime)*, 2015, pp. 1–12.

[13] Z. Dong, K. Kane, and L. Camp, "Detection of rogue certificates from trusted certificate authorities using deep neural networks," *ACM Transactions on Privacy and Security (TOPS)*, vol. 19, p. 5, 09 2016.

[14] Cisco Talos Intelligence Group, "Phishtank," n.d. [Online]. Available: https://phishtank.org/

[15] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019. [Online]. Available: https://doi.org/10.14722/ndss.2019.23386

[16] Anti-Phishing Working Group, "The APWG eCrime Exchange (eCX)," https://apwg.org/ecx/, n.d., accessed: 30-09-2021.

[17] M. Wullink, M. Muller, M. Davids, G. C. M. Moura, and C. Hesselman, "Entrada: enabling DNS big data applications," in *2016 APWG Symposium on Electronic Crime Research (eCrime)*, 2016, pp. 1–11.

[18] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, Oct. 2015, pp. 542–553. [Online]. Available: https://doi.org/10.1145/2810103.2813703

[19] "scikit-learn: machine learning in python — scikit-learn 1.0 documentation," https://scikit-learn.org/stable/, (Accessed on 10/15/2021).

[20] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, R. Khayami, K.-K. R. Choo, and D. E. Newton, "Drthis: Deep ransomware threat hunting and intelligence system at the fog layer," *Future Generation Computer Systems*, vol. 90, pp. 94 – 104, 2019.

Explainable Artificial Intelligence to Enhance Data Trustworthiness in Crowd-Sensing Systems

Sam Afzal-Houshmand*, Sajad Homayoun*, Thanassis Giannetsos‡, Christian D. Jensen*
*Technical University of Denmark (DTU), Cyber Security Section, Denmark
‡Ubitech Ltd., Digital Security & Trusted Computing Group, Greece
Email: saaf@dtu.dk, sajho@dtu.dk, agiannetsos@ubitech.eu, cdje@dtu.dk

***Abstract*—Around the world there has been an advancement of IoT edge devices, that in turn have enabled the collection of rich dataset part of the Mobile Crowd Sensing (MCS) paradigm, which in practice is implemented in a variety of safety critical applications. In spite of the advantages of such dataset, there exists an inherrent data trustworthiness challege due to actors such as malevolent input. For this issue there have a significant quantity of proposed solutions based on the usage of conventional machine algorithms towards the purpose of sifting through faulty data without any assumptions on the trustworthiness of the source. In this context there is a number of open issues such as how to cope with strong colluding adversaries while simultaneously efficiently managing the sizable influx of user data. Previously, these open challenges have attempted to be addressed by the use of deep learning schemes which works as a black box and conventional machine learning classifiers for detecting and filtering out false data point. This framework provided a considerable improvement in addressing these challenges as compared to conventional machine. However, what is the next step in achieving improvements upon those results. Where do we go from deep learning? In this work we propose that the usage of explainable artificial intelligence (XAI) may provide even more efficient performance as we refine the conception of the models moving away from a black box and will be actual able to comprehend how we end up with certain outcomes in terms of accuracy in the presence of adversaries and what can be done to adjust this to an optimal desired outcome. To this end we provide a prototype implementation coupled with a detailed performance evaluation under different scenarios of attack employing both real and synthetic datasets. Our result suggest that the addition of XAI introduce a comparative improved performance outperforming other schemes while improving upon existing schemes.**

***Index Terms*—Mobile Crowd-Sensing, Explainable AI, Adversarial Machine Learning, Data Trustworthiness, Timeseries analysis.**

I. Introduction

Temporal data can be quite unpredictable to humans as it can be hard for us to understand and explain why changes over time happen by just observing the data as an instance like a picture or texts. That is why it can be tricky to represent temporal data as a signal that varies as a function of time for us due to the ubiquitous nature of temporal data. To leverage underlying information of such data we need to leverage additional methods along with expert knowledge [1].

Abundances of devices worldwide have created a mobile sensing landscape that requires users to feed information about the sensory data of the environment, all within an sensing paradigm known as mobile crowd sensing (MCS) [2] which often takes the form of a temporal data set. In terms of security there is pertaining issues to data quality and integrity since it is a very open platform [3]. Previous works in the area toward a solution have used solutions based upon Machine Learning systems to gain insights to fixing those issues. ML systems are trained using MCS datasets that are assumed true and benign whereas maliscious actors can affect those systems negatively by targeting training data through simple or advanced manipulation or directly forcing the models toward a desired output which classify incoming events.

Other works in this area have attempted to address security issues using adversarial machine learning leveraging conventional ML based techniques toward distinguishing between malicious data from benign data [4]. However, those conventional techniques have proven to be inefficient especially when addressing pertaining issues with IoT data associated with concept drift which can be cumbersome as it related to feature extraction. Works have suggested and investigated how concept drift along with falsified information caused by many untrustworthy data sources within a distributed environment such as MCS can have an impact on the accuracy of the system across multiple levels, all within clustering and classification processes while intentionally (or unintentionally) being masked by intelligent attacker or malfunctioning users. We have previously furthered these investigations by looking into applying advanced Deep learning schemes [5] that were capable of distinguishing malicious and benign data without any assumptions regarding the trustworthiness whilst proving robust against many colluding intelligent adversaries. In this paper, we take all this a step further by addressing the plausible relation between the sensory input from devices over time via the application and integration of explainable artificial intelligence (XAI). This is the aforementioned additional methods that we use to gain knowledge and insights regarding our temporal data set that can be typically restrictive for human representation [1]. Using a similar pipeline as with FSD [5] we further expand upon it to be more exhaustive with our investigation while introducing the aspect of XAI to create the XFSD framework.

II. Related Work

A lot of papers showcase the work done within the field of applying Machine learning and Deep Learning (DL)in

the mobile crowd sensing infrastructures, addressing issues pertaining to concept drift, data integrity, sparse sensing [3] [6] [7]. In this paper, we focus on work that applied XAI to sequential time-series data so we can possibly incorporate it into to the existing FSD framework [5] [8].

There is a great variery of works that apply XAI to time-series information but it can be broken down into its most basic general categories of convoulutional neural network based methodologies, recurrent neural network based methodologies and non-sequential based methodologies [1].

Generally, RNN are typically very applicable to time-series datasets which also incompass the application of LSTM model that retains the whole history of the time data. The XAI methodology used typically within this context is some form of SHAP which is a model-agnostic explanation method. Kim et.al where the first to suggest using SHAP algorithms to explain the output of RNN used on time series data to highly accurate outcomes. SHAP has been highly successful for financial time series information prediction and explanation [9] [10] [11] [12].

Within the category of CNN based methods, the prevalent approach of applying XAI to time-series seem to be class-activation mapping. This a post-hoc method that provide explanation that highlights regions in the input data which affects the CNN ouput (or any other DL model applied to time-series) as shown across the work of multiple groups [13] [14] [15] with highly accurate results. Within the category of CNN there is more interesting approaches such as ConvTimeNet which occludes part of the time series and compute the probability of the predicted class [16]. Another approach is Gradient*Input with attempt to identify the contribution of the input raw data when performing time-series classification [17].

In terms of non-sequential methods used for time-series explanations there is the methodology of generating explanation using LIME to lesser high accuracy in regards to the sequential explanatory methods but they are highly efficient [18] [19] [20].

In terms of how to implement XAI numerical values as a means to improving accuracy there is sparse work to be found in the area of MCS. Typically, XAI values are used a a way of visualizing some sort of point that explains the model in question, however, methodologies that actual makes practical use of XAI values can found such as using numerical XAI values as a supplementary value for classification purposes [11] [21].

III. Towards Trustworthy MCS Tasks

Mobile Crowdsensing (MSC) information platforms allows users to contribute during its sensing process by uploading their contributions to a central server where the collected information can be processed by local analytic algorithms with the purpose of generating data consumable to requesting applications in different fields such as medicine, traffic flow etc. [22]. To put it mathematically, it is denoted for a specific time window with n time steps and m sensors, we gather a dataset D containing a sequence (S) for each sensor j where $S_j = [v_{1,j}, v_{2,j}, \cdots, v_{i,j}, \cdots, v_{n,j}]$. In this denotation where $v_{i,j}$ represent the value of sensor j at time step i, it can be extrapolated that sequential analysis techniques are beneficial in possibly extracting usable insights from the relationships between the various sequential feature values. D in (1) shows how S sequences build the matrix as the dataset.

$$D = \begin{bmatrix} v_{1,1} & \dots & v_{1,j} & \dots & v_{1,m} \\ \vdots & & \vdots & & \vdots \\ v_{i,1} & \dots & v_{i,j} & \dots & v_{i,m} \\ \vdots & & \vdots & & \vdots \\ v_{n,1} & \dots & v_{n,j} & \dots & v_{n,m} \end{bmatrix} \tag{1}$$

The sequential structure of the data makes sequential analysis techniques like LSTMs more interesting as LSTMs are beneficial in extracting useful insights from the relationships between the various sequence values.

A. Threat modelling

Colluding adversaries have the objective of negatively affecting MCS application by applying an adversarial agent that attempt to make malicious measurement values seem legitimate to the application. With the established denotation this can be considered as an adversary may change the input value $v_{i,j}$ in S_j to $v'_{i,j}$, where $v'_{i,j} \neq v_{i,j}$ to maximize the distortion "$\max\{|v_{i,j} - v'_{i,j}|\}$", where the distortion should be lower than a maximum allowed pre-determined by the adversarial actor.

There is general consideration of two primary adversarial attack models which have many offshoots that may also be taken into consideration for such a platform [4] i.e.: 1) pre-training (poisoning) attacks, and 2) post-training (evasion) attacks. For the sake of consistency we refer to data poisoning as being the pre-training attacks and evasion attacks as post-training attacks. With pre-training attacks the adversaries attempt to inject malicious data in an attempt to poison the training dataset in training time and, thus, decrease the classification accuracy of the classifier. Furthermore, in the area of data poisoning it is possible to apply more advanced attack modems such as the one shown in the work of Miao et.al. where there is consideration of how to optimize the parameter values for two specific data poisoning attack strategies i.e. target attack and availability attack [23].

The evasion attack are referred to as the post-training attack strategy which happens in testing time. With the post-training attack model, the adversaries objective is to mislead already trained classifiers to misclassify samples towards a malevolent intent.

To illustrate the adversarial objective, let us assume $f(x_i) = y_i$ as the mapping function to calculate x_i to y_i. For every new sensed values x'_i, f gives a new output $f(x'_i) = y'_i$, and we have the following cases:

- *True Positive*: if x'_i is positive and f correctly outputs positive, there is no loss on the application.
- *False Positive*: if x'_i is negative and f outputs positive, there is a loss ϵ on the application.

- *False Negative*: if x'_i is positive and f outputs negative, there is a loss l on the application.
- *True Negative*: if x'_i is negative and f correctly outputs negative, there is no loss on the application.

Principally, with a machine learning technique we try to minimize $|f(x'_i) - y'_i|$ which means minimizing False Positives and False Negatives. Contrarily, adversarial attackers attempts to maximize the impact of the attack by maximizing $|f(x'_i) - y'_i|$. For the remainder of the paper we refer to adversarial data as positive class of data, and legitimate data as negative class.

IV. Explainable AI

In its base principality, it can be said at Explainable AI (XAI) is AI where the results of the intelligent solution in interpretable which is in contrast to the typical black box architectures followed by some machine learning algorithms especially neural networks. Machine learning has a long history of being applied to time-series data, however, the high complexity of those solution tends to make the interpretation difficult. Without a quantitative assessment it may be hard to understand areas of temporal data which may affect the prediction. Enter XAI which may offer a solution to these issues, adding the additional knowledge that may strengthen the overall model essentially aiding the models prediction by additional information [1] [24] [18]. There are a multitude of algorithms to use when applying explainable AI on time series information.

As this paper deals with sequential time-series data that are found in MCS contexts, we are focusing on two of the most common XAI algorithms in their base forms focusing on XAI in timeseries, namely SHAP (SHapley Additive exPlanations) and Class Activation Mapping (CAM). Moreover, we will study the impact of a well-known non-sequential based XAI algorithms called LIME (Local Interpretable Model-agnostic Explanations) to study the difference between sequential and non-sequential XAI algorithms in detecting malicious samples generated by adversaries.

Generally, the pipeline of applying XAI to a time-series such as the data in MCS can denote numerical values as a scalar from -1 to 1 signifying importance score or impact. To break this down simply we take the model and the raw data as input and obtain a vector of fixed length determined by the sliding-window in which we train the model. The significance of the elements within what the vector consist of represent the impact the values at a given time-frame have on the prediction. So this numerical represent whether a given step in the raw dataset affect the prediction negatively or positively (and how much with 1 being the maximum impact) dependent on a baseline computed by using the model and raw data as input with the resulting prediction which can be used for comparison. This way XAI is applied holistically to the model and will explain what kind of importance areas of the input stream has for a given feature which are values that can be used in a variety of ways such as enrichment or weight adjustment for the NN model. The properties of these values will add possible enrichment as one of their inherent capabilities is that they are also affected by adversarial attack strategies which can make these values distinguishable dependent on what type of attack has been applied to them

For SHAPley we are calculating the impact an value has on its prediction (meaning it is independent from other categories). In time-series we calculate SHAP values by a window of values which can be aggregated together for a singular/shortened representation. The resulting vector is concatanated with values for each category feature so we end up with a series of vectors where each section of elements represent a feature explanatory value for a given time-frame computed from the information of the time-frame values and prediction model.

In cases of univariate sqequential data it may be advantagous to just use the XAI values without applying any aggregation (some dimensionality reduction can be applied for very sparse cases) to retain the ordinal info and get a good representation of the impact (denoted as score) over time. To that end, an sliding-window of a given size can be applied so we end up with a vector of a fixed size representing the XAI values for that chunk of samples. A lot of approaches aggregates these chunks to reduce the dimmensionality.

A. SHAP (SHapley Additive exPlanations)

Shapley values can be summarized as the average of the marginal contributions across all permutations of a machine learning model. There are multiple advantages of SHAP. One of the advantages provided by SHAP include global interpretability which is the collective SHAP values showing how much each predictor contributes be it negative or positive to the target variable. The second benefit is the local interpretability where each observation gets its own set of SHAP values. Third the SHAP values can be calculated for tree based models [1].

Once the DL model has been run on with a dataset we run the DeepExplainer (SHAP function that takes data and model as input) and extract the SHAP importance score values locally [9]. So once we have a model from a collection of observations we can feed this into the explainer API for SHAP and extract some numerical values showcasing the importance of each feature [25] [26]. In context of time-series SHAP calculate shapley values step-wise in a time series. This is also called permutation importance which is an approach that extracts importance per permutation enabled by XAI. SHAP based on single feature orderings which is approaches already proposed for (Saabas et.al) take this single feature ordering done step-wise in frames of time samples within a time series where it effectively provide a combination of applying XAI to time-serie by calculating impact values step-wise for each time-step applied to a single feature ordering (often called and approximate method with the TreeSHAP algorithm). In simple terms, the resulting value from this variation of SHAP is a scalar from -1 to 1 that can be interpreted as an impact score the feature value of a given feature at time i have on the prediction of said feature derived from the pre-trained DL model applied on the data if it took some baseline value based on the whole history of the predictions. From the

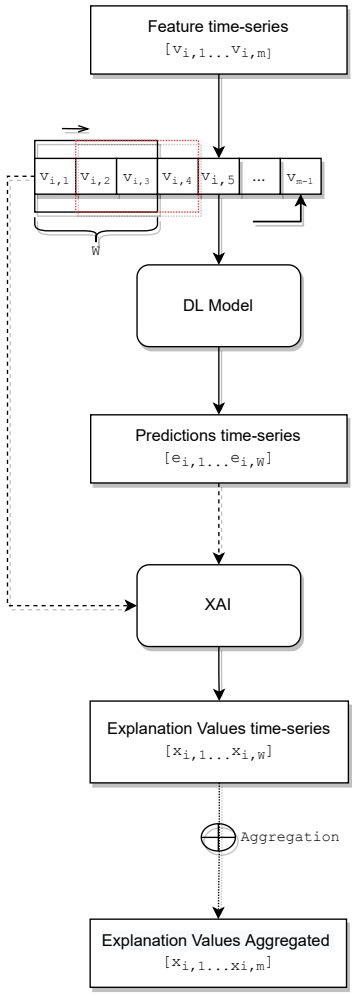


Figure 1: . A simplified version of the general XAI pipeline. The concept is that using the time-series feature values and applying a sliding-time window to feed into a LSTM model to gain prediction we can calculate evaluations scores/impact scores which map the impact a feature value has on the prediction at a given time as determined against a baseline calculated from all predictions denoted from the trained model for the given sequence of size s. That way we obtain evaluation values in the form of vectors of fixed length. Now we can apply an aggregation model to get a summation of these sequence values if needed.

SHAPley software description the prediction starts from the baseline. The baseline for Shapley values is the average of all predictions. There is also other method of gaining the baseline such as an approximate method suggested by Saabas. The importance of the SHAP values are thus determined by the whole history of the RNN/LSTM model used for prediction and calculated for each step providing importance of each element in relation to the whole sequence in question for a time-frame and total sequence of the whole model post-hoc. This means that the importance value are does not signify the importance of a given time-step in relation to other time-steps in a given time-frame but it is the importance of the time-step in relation to the total prediction model for a given time-frame.

B. Class Activation Mapping (CAM)

Unlike the SHAP which was built in relatio to recurrent neural network model/ Long short-term memory the Class activation mapping (CAM) was build in regards to Convulotional neural network model. CAM has the purpose of highlighting the regions of the input data with the most influence on the CNN models output classification prediction. This is reliant on the prescense of a Global Average pooling at the end of the CNN layers [17]. Those features which will be obtained are then fed to a fully connected layer with softmax activation which produce an output that is interpretable i.e. an importance score. CAM values denotes how much contribution the time segment has on the class in question based upon the DL model and raw data input. Therefore, in this case the importance score is calculated as related to an baseline of sorts from the prediction and the input signifying the importance of certain areas of the sequential data as it relates to one another given by the resulting predictions [27]. In the end the input and outcome of CAM and SHAP in context of time-series is the same, where the values in both cases signifies an importance score. However, with CAM this is computed in relation to other values in the same time-frame whereas SHAP is calculated based on a general baseline. As this also indicates CAM necessitates that a CNN model has been applied. Unlike with RNN and SHAP we do not need to fully intialize the model once before performing the explainer importance score extraction [1] [13] [14] [15] [28].

To distinguish between the importance scores calculated by CAM and SHAP it can be stated that in this context of time series that CAM denotes importance scores that signifies the importance of regions of values in a time-frame as it relates to other values in that given time-frame (also called sub-sequence) for the predicting model. SHAP calculate the importance of each values in time-frames signifying the impact of the given value toward the prediction in totality for a given time-frame and not its importance in relation to other values in its timeframe i.e. the importance of the given value is determined by whole sequence of the prediction model in the time-frame. Essentially, this means that we are at risk with SHAP to end up vectors of importance scores where the standard deviation is very small (can also be very large) whereas with CAM we will always have vectors signifying

areas which are more important than others in input sequence towards predictions resulting in consistently large standard deviation in values.

C. LIME (Local Interpretable Model-agnostic Explanations)

Another approach which we already took inspiration from with the SHAP approach is the non-sequential approach using LIME (Local interpretative model-agnostic explanations). LIME explains the prediction of any classifier in an interpratative manner by learning an interpretative model locally around the prediction. LIME does not consider the sequential order of data so a snapshot is suficable. It is however more consistent. So in this case the coalition that is signified is how one given value within a time-frame correlates to one given prediction value. Otherwise, a similar pipeline with numerical importance scores was used as with CAM and SHAP. [18].

V. XFSD Conceptual Architecture

The base Fake sequential data detection (FSD) framework [5] was created to benefit from the relationships between samples of different time orders so as to accurately predict the values of the next time step to higher degree of success of similar attempt with conventional machine learning in the area. That system utilized distance metrics for comparing the predicted data and real data, at each time step, in order to prepare samples for a one-class classification approach essentially drawing a boundary around the allowed deviations between predicted and real data values.

FSD takes advantage of sequential relationships, between collected data instances, by combining the use of LSTM, as a deep learning technique, and one-class classifier which captures characteristics of the training data input towards identifying instances of a specific class amongst all instances.

With the XFSD we add in the numerical values from Explainable AI (XAI) models which represents the importance of each feature over time. The input for XAI is the raw sequence of values for each feature (sensor) along with a trained model which then output a numerical vector that denotes the importance score of the feature for a time-frame in the end which can then be concatenated into a vector.

Like most deep learning frameworks our approach consists of two main phases, namely the *Training* and *Testing* phases. In the *Training Phase*, we make a predictor for each sensory data as well as a final one-class classifier along with a XAI value generator taking the values and model as input whereas during the *Testing Phase*, we feed a combination of malicious and benign testing samples in order to evaluate how good our trained models behave in separating data generated by adversaries.

A. Training Phase

Figure 2 shows the steps followed by XFSD to train the final one-class classifier to distinguish benign samples (not generated by adversaries). Similar to FSD [5], XFSD framework trains a sequence predictor (P_j) for every sensor j that takes input sequence and predict the next step value (Step 1 of Figure 2 and Algorithm 6). Step 2, for each timestep, uses the trained predictor (P_j) to estimate the expected value (e_j) at time step i by feeding the sequence (S_j) of all values of sensor j from timestep 0 to timestep $(i-1)$. α_i, then, are the distance vector of size K that is computed by the K distance metrics (e.g. Cosine Distance, etc). At the same time, feeding the sequence S_j and the predictor P_j will to X_j will generate a vector of values ($\beta_{i,j}$) as the explanation of the data at the current step. Concatenating and flattening α_i and all $\beta_{i,j}$ will give $\vec{Z}_i$ vector that would be row i of Z matrix shown in (2). Thereafter, the prediction are compared to the corresponding real value where the distance is computed between the two and fed to a one-class classifier. However, before feeding the vector of distance measures we create corresponding XAI values which represent the importance score for a given a feature in time-frame as described in the background knowledge. The input for the XAI (where the algorithm should be interchangeable) is the raw data and the model wherein the XAI functionality will output corresponding importance scores which is concatenated with the original vector of distance vector. Essentially, we end up with a vector of the distance measures concatenated with a vector of multiple vectors of same length concatenated for each feature. This is then fed to the one-class classifier.

1) Step1: Train timeseries predictors: This step trains a predictor (P) for each sensor that is able to predict sensor value of timestep i based on the sequence of values at previous timesteps. Although this step is not limited to sequential based algorithms, we recommend using Recurrent neural networks (RNNs) and Convolutions neural networks (CNNs) as they are widely used in predicting timeseries. The output of Step 1 (*Predictors*) will be used later in Step 2.

Algorithm 1: STEP1: Train timeseries predictors

Data: *TD* as training dataset
output: trained *Predictors* for timeseries

```
Predictors ← empty list;
for j = 1 to m do
    S_j ← sequence of all values for sensor j;
    P_j ← Train_TSP(S_j);
    Predictors.add(P_j);
return Predictors;
```

2) Step2: Explainable feature extraction: We apply an interchangeable XAI algorithm process which effectively calculates a numerical representation of the important areas of the data input based upon the sequential model predictor. Step 2 of Figure 2 shows the procedure of computing $\vec{\alpha}$ and $\vec{\beta}$ into $\vec{Z}_i$. The output of this step is matrix Z that will be used by Step 3 to train the final one class classifier.

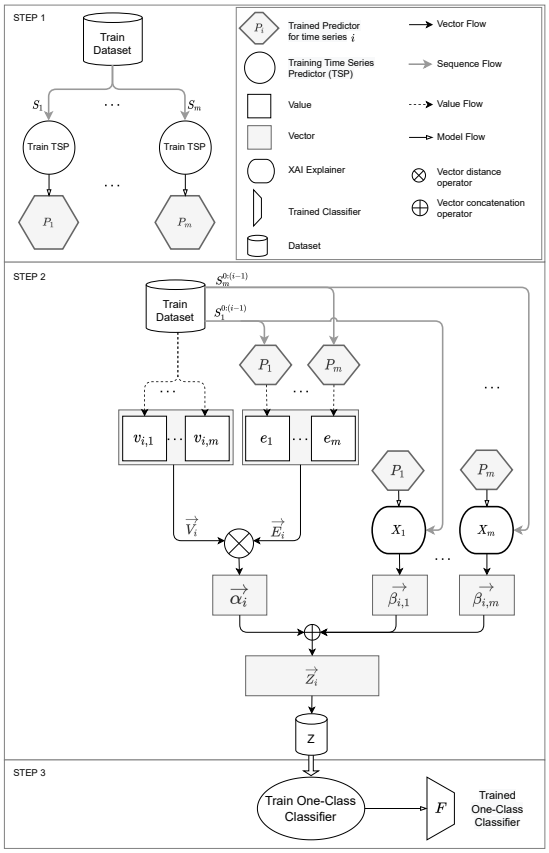


Figure 2: Training phase of XFSD

$$Z = \begin{bmatrix} z_{1,1} & \dots & z_{1,k} & \dots & z_{1,q} \\ \vdots & & \vdots & & \vdots \\ z_{2,1} & \dots & z_{i,k} & \dots & z_{i,q} \\ \vdots & & \vdots & & \vdots \\ z_{n,1} & \dots & z_{n,k} & \dots & z_{n,q} \end{bmatrix} \quad (2)$$

where $q = K + (m \times a)$ in (2).

3) Step3: Training one-class classifier: Succeeding step 2, all legitimate data is fed to the *Predictors* where the system will calculate Z for each time step i containing all *allowed deviations* between the predictions and the real data plus XAI values for all time steps. Z matrix is suitable for a one-class classifiers as it is achieved by feeding only benign data to the predictors and the explainers. The one-class classifier algorithm attempts to find a boundary around the training samples (achieved from benign samples) in order to separate them from the data from other distributions (line 1 of 2, where F in Figure 2 would be the trained one-class classifier.

Algorithm 2: Step2: Explainable feature extraction

Data: *TD* as training dataset
input : *Predictors* as list of timeseries predictor
output: Z as dataset of deviations and XAI values

$Z \leftarrow$ empty matrix;
$\Theta \leftarrow K$ distance metrics;
for $i = 1$ *to* n **do**
  $\vec{V}_i \leftarrow$ [vector of values at timestep i of TD];
  $\vec{Z}_i \leftarrow$ empty vector;
  for $j = 1$ *to* m **do**
    $P_j \leftarrow Predictors[j]$;
    $e_{i,j} \leftarrow P_j(\vec{V}_{i-1,j})$;
  end
  $\vec{E}_i \leftarrow [e_{i,j}$ for $j = 1$ to $m]$;
  $\vec{\alpha}_i = [\Theta_1(\vec{V}_i, \vec{E}_i), \dots, \Theta_K(\vec{V}_i, \vec{E}_i)]$;
  $Z_i.concatenate(\vec{\alpha}_i)$;
  for $j = 1$ *to* m **do**
    $P_j \leftarrow Predictors[j]$;
    $S_j \leftarrow$ sequence of all values of sensor j from timesteps 0 to $(i-1)$;
    $\vec{\beta}_{i,j} \leftarrow X[j](S_j, P_j)$;
    $\vec{Z}_i.concatenate(\vec{\beta}_{i,j})$;
  end
  $Z.add(\vec{Z}_i)$;
end
return Z;

Algorithm 3: Step3: Training one-class classifier

Data: Z as the training dataset
output: F as trained classifier

$F \leftarrow$ train_occ(Z);
return F;

B. Testing phase

Figure 3 shows how XFSD works in real time to detect adversarial samples. For each testing time step i, XFSD uses predictor P_j to produce e_j for sensor j, then it concatenate and flatten $\vec{\alpha}$ and $\vec{\beta}$ into $\vec{Z}_i$ ready for the one-class classifier F to specify if the data received in the current timestep follows the distribution of benign data. As F is a trained one-class classifier on legitimate deviations between real and predicted values as well as the XAI values, the output dictates whether $\vec{Z}_i$ is legitimate or not. If F say $\vec{Z}_i$ is not in the distribution of benign samples, then $\vec{V}_i$ will be flagged as malicious as one or more sensor values at timestep i in $\vec{V}_i$ were generated by adversaries.

VI. Experimental Setup

XFSD is an extension of FSD with the addtion of XAI numerical values as a way of improving distinguished. Simultaneously we add more advanced attack strategies to supplement the investigation of the effect of XAI-FSD.

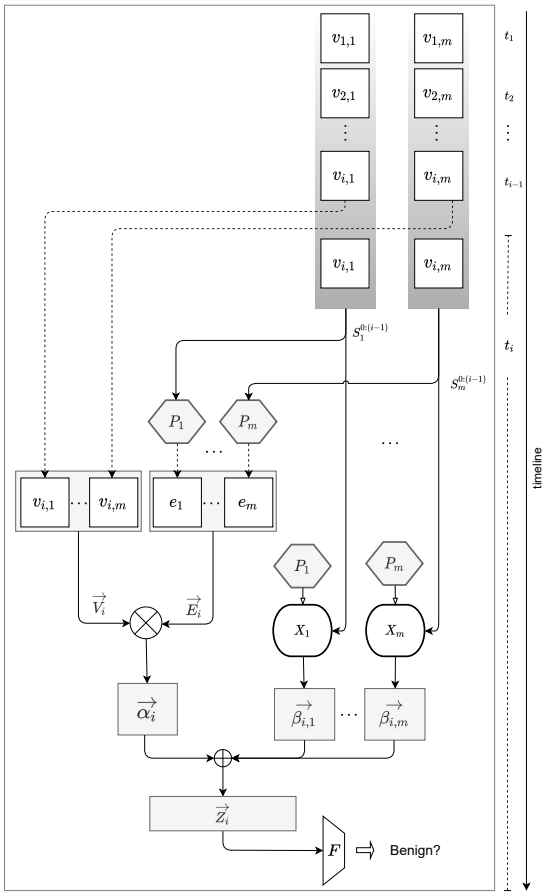


Figure 3: Testing phase of XFSD

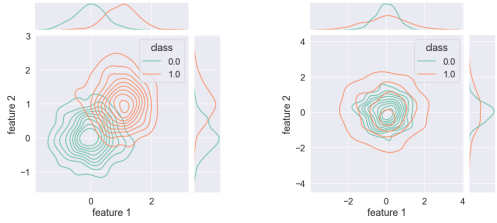
A. Adversarial Behavior

As with FSD, the XFSD is a framework for characterizing between inlying and outlying sensory data reports in the presence of adversarial malicious users, where we assume that colluding adversaries are attacking the data collection process (poisoning attack) or the classification process (evasion attack). Essentially, the adversary may attempt to feed malicious data points which are then included in the training of the deep learning model used for the classification in XFSD or sends malicious data points *after* the model and classifier has been trained on the real benign data points. Therefore, we categorized attack strategies into pre-training and post-training attack strategies respectively. In the addition we have furthered the attack strategies in the pre-training (data poisoning) with advancement from bi-level optimization. The performance of the framework was evaluated based on different levels of distortion by colluding adversaries trying to impose on the data trustworthiness. In XFSD we consider two main attack approaches for each attack strategy: *1) distribution attacks (Attack Cases I & II)* and (Attack Cases VI & VII), which manipulate mean or standard deviation to generate adversarial samples which are different from the benign ones; And *2) position attacks (Attack Cases III, IV & V)*, which manipulate the position of injecting adversarial samples in the sequence of values which in turn targets the order of samples. Position attacks can only be applied after a distribution attack toward changing the order of samples to be injected in the sequence.

Attack Case I: The adversaries may affect the system uncertainty about the true value of the sensory data by setting $\sigma' = \sigma$ and $\mu' \neq \mu$ which represents a malicious standard deviation equal to the standard deviation of the legitimate data points with smaller/larger μ as Equation (3) where λ is a scalar value as the deviation factor. Adversarial samples generated by $\lambda > 2$ are not attractive in our data as concluded with base FSD, as they are easier to detect due to their lower overlap with benign samples, so we limit λ between 0 and 2 to study more realistic data distributions.

$$\begin{cases} \mu' = \mu + (\lambda \times \sigma) & 0 < \lambda \leq 2 \\ \sigma' = \sigma \end{cases} \tag{3}$$

To further illustrate what the actual structure of this attack we refer to Figure 4a where there is a comparison of the distributions of legitimate samples and adversarial samples generated by Attack Case I for a simple dataset with only two features, where $\lambda = 1.0$. The adversaries may choose the λ value depending on the attack strategy (pre-training or post-training) and the number of attacking samples they want to inject/test notated as a percentage of the total number of samples. As seen in the figure, changing the mean value would have a greater impact on changing the distribution as it will not have a huge overlapping region with the actual distribution of the legitimate data. The intuition is that trained classifiers should actually work better in detecting malicious data if they were trained solely on a legitimate distribution.



(a) (b)

Figure 4: Distributions of legitimate samples (class 0) vs. adversarial samples (class 1) for two features with $\mu = 0$ and $\sigma = 1.0$; (a) Attack Case I, and (b) Attack Case II.

textbfAttack Case II: Adversaries may affect the system uncertainty by selecting an adversarial distribution based on equation 4 where the adversary's goal is to keep the same mean but changing the standard deviation.

$$\begin{cases} \mu' = \mu \\ \sigma' = \sigma + (\lambda \times \sigma) \quad 0 < \lambda \leq 2 \end{cases} \tag{4}$$

Similar to with case I we refer to the similar Figure 4b where it is shown that the distributions of a dataset with legitimate and adversarial data from Attack Case II with $\lambda = 1.0$. This attack case is designed to better reflect the real-world case scenarios where adversaries attempt to gradually change the classification behaviour by performing concept drifting by modifying the standard deviation over time.

Attack Case III: In this case of a positional attack strategy, the adversaries may affect the system uncertainty about the true values by selecting a different distribution by changing the *order* of legitimate and malicious data points in the sequence of data. In this case all legitimate data comes before malicious data points in the sequence.

Attack Case IV: The opposite of III, in this case all malicious data come before benign data points in the sequence.

Attack Case V: The adversaries attempt to affect the system uncertainty by putting malicious batches of data in between legitimate data batches. Variations of this served as inspiration for the expansion of positional attacks including more specific patters including randomization, normal and different sized windows of malicious data batches inserted.

Attack Case VI: The availability attack aim is to maximize the error at the point where the adversaries are sifted by the XFSD. Essentially the attack aims to deviate the outcome of the truth as much as possible creating as many false-positives from true-negatives as possible i.e. the attack attempt to make the system accept as much data as possible malicious data points included. Practically, this means that we feed the system a dataset during the pre-training which opens it up as much as possible before testing phase in terms of what boundaries the classifier can draw for benign samples putting a focus on larger variance in the data poisoning process. With the bi-level optimization approach we can gain the optimal attack strategy i.e. the optimal shift, distribution, the number of malicious samples which deviates the outcome as much as possible from the truth. Within this case we consider both case I and II i.e. shifting the μ and σ.

Attack Case VII:Target attack focuses on skewing the estimated truths of the target objects to certain target answers pre-determined by the attackers through poisoning the sensory data. So if the truth is determined to the attackers target we succeed whereas with the availability attack we try to deviate that response as much as possible and not to a specific answer which is taken into account when developing the optimized solution to be used by the attacker. Practically, we generate a dataset to a specific pre-determined target (decided arbitrarily for demonstration sake) and feed it during pre-training and observe the effects by testing phase. What the target is could be dependent on the desired outcome of the adversary e.g. denial of service, delay etc. Similar to availability attack we obtain the optimal manipulable variable to obtain the outcome which fits to a pre-determined target by a bi-level optimization. Within this case we consider both case I and II i.e. shifting the μ and σ.

VII. Results

–In this section, we evaluate the performance of XFSD in the presence of strong adversaries under two major attack strategies (denoted as pre-training and post-training). For that purpose we utilize F-measure as the evaluation metric since it considers True/False positive rates, recall and precision. We divided the dataset into two subsets with a 60%-40% split. It ought to be mentioned that redundant results have not been reported for the sake of saving space in the paper.

Furthermore, for cases VI and VII we have considered only pre-training where the purpose of those cases was to evaluate how well our XFSD framework holds up to more advanced poisoning attacks as compared to the poisoning attacks in cases I and II making it a comparison between cases essentially.

Firstly, we focus on distribution attack cases (i.e. I and II) followed by a section with positional attack (i.e. attack cases III, IV and V) succeeding it which also dives into the expansion of the positional attack patterns introduced with XFSD. After that, we have a section for the advanced attack cases (i.e. VI and VII) where both cases include manipulation with both μ and σ (i.e. attack cases I and II) in context of more advanced pre-training (data posioning) attack strategies as described previously in this paper.

We reviewed the performance of different scenarios of incorporating the XAI values to the base FSD framework by exploring various utilization's of XAI values. In this case we refer back to our overall architecture for XFSD and explore the impact of incorporating numerical values denoted by an XAI algorithm in different approaches before moving on to investigating different types of algorithms effectively adjusting the identified key parameters of sliding-window size and aggregation algorithm.

We investigated multiple ways of including XAI values where we focused of adjusting the parameters of sliding-window size in order to retain ordinal information of the sequential data. This included enriching the vector at time T_i with the XAI values from $T_i - Sliding - Window_Size$ where we apply various time-window length of 2, 100,1000,10000 time-steps to the DL model so then with the XAI we end up with fixed length vectors that can be concatenated with the distance metric vector which is obtainable by base FSD without any XAI applied. During our experimentation we observed that a sliding-window of a 1000 denoted the best results in terms of highest measured f-measure. After a sliding-window size of a 1000 the accuracy started decreasing. Hence, forthcoming in our results section we report the scenario where we used a sliding-window size of a 1000 and concatanated the XAI values directly with the distance metric vector as they are of fixed length. Another variation of this was to aggregate the vector of XAI values to reduce dimensionality in the final concatanated vector but this has been separated to its own experimentation in this paper as to also investigate its impact and different aggregation methods. Thus with the

optimal setup for incorporating XAI algorithms established we report the results as to do the comparative analysis of applying various XAI algorithms along with the base without those values i.e. SHAP, CAM and LIME.

A. Results of XFSD Distribution attacks

We are trying to establish the improvements and accuracy that can be achieved by applying quantifiable values derived from XAI algorithms on time series information fed to the base FSD framework. Previously we have established the accuracies possible with deep learning and conventional machine learning. In the results section we go through the results of each XAI algorithm along with the FSD without XAI values for each case as to make the comparative performance easier to comprehend. Furthermore, we apply this for all cases in all strategy scenarios.

1) XFSD case I and II in pre-training attack strategy: With the pre-training attack strategy the adversaries tries to poison the training datasets manipulating the data based on an attack vector. It is generally harder to detect malicious samples in the pre-training than the post-training since classifiers have been trained to recognize the adversarial samples as legitimate. Increasing the rate of adversaries in pre-training has a higher impact on classifiers as it consider all or part of the injected samples as noise effectively changing the perception of the classifier. For the same reason an attacker would chose a higher deviation factor as the end-goal is to mislead the classifier's perception in that scenario. In XFSD we also test the comparative performance of different algorithms for XAI usually denoted as numerical values concatenated to FSD distance metric values along with the overall impact of applying XAI values to the FSD framework.

In table I and II the results for attack case I and II is shown for the pre-training attack strategy. Each row of the tables show F-measure for a specific λ value where the attacker generate malicious data using the aforementioned attack vectors. From what we know from theorem increasing the λ ought to decrease the adversarial data rate, leading to less accurate classification accuracy in the pre-training strategy as the concept is to train the classifier to detect wrongly i.e. skew the perception of the classifier. Increasing the adversarial rate in pre-training prevents the classifier from ignoring the impact of malicious data samples in the training which means more malicious samples results in a more misled classifier. This degradation tendency is also reflected in the results from said tables. All of these reflections where as expected from what we know from FSD without the XAI values. The interesting aspect is the impact of applying different XAI methods. As showcased in the tables the tendencies are more less the same across different algorithms in pre-training which makes for a very simple interpretation as the impact of different attack cases in pre-training strategies is the same regardless of algorithm applied. The only difference is the scale of accuracy between different XAI methods under different cases. Generally, the introduction XAI to base FSD provide a proportional improvement in terms of performance denoted by F-measure meaning that the degradation is the same in regards to different configurations of adversarial attacks in pre-training attack strategy between FSD and XFSD, however, the F-measure is scalably larger meaning that there is an improvement in the performance when adding XAI values to the regular value vector which present an improvement for the classifier. This confirms the expectation we had to introducing XAI values as the additional information from the explanations improves the overall performance of the classifiers. A valuable property of XAI is that the values that they denote are equally affected by adversarial machine learning. This means that the introduction of XAI provide values for the classifier that are differentiable with or without manipulation from adversaries. This in turn is advantageous for the one-class classifier as they will receive enriched datasets more elaborate (more values) and therefore distinguishable between samples.

The reasoning behind the improved performance with XAI is that adversarial attacks strategies also impact explanation values/importance score besides the general enrichment of more information over a time-frame that is obtainable by adding XAI values. If we have the importance value they enable us to detect that impact, essentially becoming an additional distinguishing value between what is legitimate and malicious. So if we have explanatory values for certain feature when it is legitimate we can measure the distance (using our defined metrics) between that value vector and a potentially malicious sample akin to what we have done for the raw values to determine the legitimacy of a sample. We have proven from theorem and observations from experiments that the explanation values are negatively affected by the adversarial strategies which is a correlation we take advantage of to provide additional information in the vectors for the classifier to utilize, hence, the addition of explanatory values create better classifiers as they have more values that can be distinguished as legitimate or malicious. Conceptually, the idea is that the explanatory values showcase the values or weights which signify the impact of a certain time-frame of data according to a trained model i.e. explanations. That overview of impacts denoted by values is affected by adversaries and can help the one-class classifier in distinguishing maliciousness by differing more than a benign explanatory distribution. Furthermore, in its foundation XAI provides values that explain the impact of input to a NN model. With that knowledge it is possible to adjust weights that may improve a model mitigating issues like concept-drift and similar or enriching datasets for classifications models by adding more values from the same distributing for the classifier to train on.

Another part of the experimentation of this project was to find the best algorithm. We had an inkling of a idea why certain algorithms may perform better given the dataset we are working with i.e. we expected SHAP to outperform the other methods as it is designed for scenarios using RNN or LSTM which is the basis of FSD which is widely recognized as the conventional way of handling time-series information. To illustrate this performance lets take an example of $\lambda = 2$ and 40% adversarial rate for all algorithms in case I in pre-

training strategy. Here the outcome is XFSD-SHAP=0.613, XFSD-CAM=0.587, XFSD-LIME=0.516, FSD=0.556. SHAP is clearly the most accurate method in terms of F-measure which is expected, however, it is interesting to note that LIME actually result in a worse performance that base FSD. The explanation for that is that LIME is a non-sequential algorithm which indicate that there is a significant amount of information that can be extracted from the sequential order of our data which is sensical since we are dealing with time-series data.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.963	0.929	0.941	0.933	0.915	0.881	0.865	0.848	0.826
	1	0.952	0.882	0.909	0.889	0.895	0.856	0.823	0.804	0.771
	2	0.914	0.837	0.876	0.849	0.828	0.819	0.792	0.769	0.719
CAM	0.5	0.886	0.841	0.8	0.791	0.777	0.741	0.725	0.707	0.686
	1	0.841	0.81	0.769	0.749	0.752	0.716	0.681	0.664	0.631
	2	0.794	0.774	0.735	0.708	0.686	0.674	0.652	0.629	0.579
LIME	0.5	0.85	0.805	0.764	0.755	0.741	0.705	0.689	0.671	0.65
	1	0.805	0.774	0.733	0.713	0.716	0.68	0.645	0.628	0.595
	2	0.758	0.738	0.699	0.672	0.65	0.638	0.616	0.593	0.543
FSD	0.5	0.843	0.797	0.756	0.747	0.733	0.695	0.682	0.663	0.642
	1	0.797	0.765	0.725	0.703	0.707	0.672	0.637	0.619	0.587
	2	0.749	0.729	0.691	0.664	0.641	0.629	0.608	0.583	0.533

Table I: Results of various XAI algorithms + FSD for Attack Case I in pre-training attack strategy

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.875	0.861	0.839	0.821	0.802	0.782	0.762	0.718	0.684
	1	0.836	0.811	0.802	0.789	0.769	0.753	0.719	0.669	0.635
	2	0.799	0.789	0.784	0.779	0.742	0.727	0.684	0.638	0.589
CAM	0.5	0.866	0.852	0.83	0.811	0.792	0.772	0.752	0.708	0.675
	1	0.827	0.801	0.792	0.779	0.759	0.743	0.709	0.659	0.625
	2	0.789	0.779	0.776	0.769	0.732	0.718	0.676	0.627	0.579
LIME	0.5	0.858	0.844	0.822	0.803	0.784	0.764	0.744	0.7	0.667
	1	0.819	0.793	0.784	0.771	0.751	0.735	0.701	0.651	0.617
	2	0.781	0.771	0.768	0.761	0.724	0.71	0.668	0.619	0.571
FSD	0.5	0.791	0.777	0.755	0.736	0.717	0.697	0.677	0.633	0.6
	1	0.752	0.726	0.717	0.704	0.684	0.668	0.634	0.584	0.55
	2	0.714	0.705	0.701	0.694	0.657	0.644	0.601	0.552	0.504

Table II: Results of various XAI algorithms + FSD for Attack Case II in pre-training attack strategy

We can easily establish that introducing certain XAI algorithms and using their values as described in XFSD provides an increase in performance of the classifier in terms of F-measure in cases I and II cases for pre-training strategy.

2) XFSD case I and II in post-training attack strategy: In post-training scenario the attacker attempt to generate samples for bypassing the classifiers during testing time without any access to the training data. Thus, for this attack strategy the models are trained on legitimate data then the overall classification is evaluated using manipulated dataset. Simply, we split our dataset into training and testing sets. In pre-training dataset we poison the training data set and in post-training we poison the testing dataset. With the post-training strategy it ought to be easier for a trained classifier to detect non-overlapping malicious samples as the classifier has the knowledge to cover the legitimate area. Therefore, adversaries will tend to not deviate the incoming poisoned dataset from the legitimate dataset i.e. not change λ much.

As shown in III and IV we can see that we achieve higher f-measures when the poisoned test data set when the maliscious samples are increased and the deviation λ is larger which makes sense since the dataset in those cases are far from the legitimate dataset making it easier to detect by the classifier. Again these are expected results for XFSD from what we have previously learned from FSD and as in the scenario of pre-training strategy the post-training strategy show similar tendencies across models for all configurations in both cases in post-training. So once again, the interesting aspect is to investigate the observable improvement and comparative behavior of different XAI algorithms, and , unsuprisingly the tendencies are the same here as they were in pre-training. XFSD-SHAP is the strongest and perform with a higher F-meaure than base FSD proving that adding XAI-values can have a positive effect as discussed throughout this paper. To illustrate, lets take an example in post-training for case II with configuration 10% and $\lambda = 0.5$. Here, $XFSD - SHAP = 0.666, XFSD - CAM = 0.677, XFSD - LIME = 0.551, FSD = 0.582$. Once again, we see the exact same tendencies comparative to one another in this scenario for these cases.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.698	0.709	0.738	0.759	0.782	0.804	0.832	0.862	0.901
	1	0.72	0.738	0.755	0.774	0.796	0.839	0.853	0.905	0.945
	2	0.729	0.747	0.763	0.795	0.815	0.857	0.871	0.939	0.986
CAM	0.5	0.647	0.66	0.687	0.71	0.734	0.755	0.782	0.813	0.853
	1	0.671	0.69	0.708	0.725	0.747	0.79	0.803	0.855	0.896
	2	0.678	0.699	0.713	0.746	0.766	0.807	0.823	0.89	0.947
LIME	0.5	0.579	0.592	0.619	0.642	0.666	0.687	0.714	0.745	0.785
	1	0.603	0.622	0.64	0.657	0.679	0.722	0.735	0.787	0.828
	2	0.61	0.631	0.645	0.678	0.698	0.739	0.755	0.822	0.879
FSD	0.5	0.578	0.609	0.623	0.633	0.654	0.675	0.703	0.723	0.769
	1	0.599	0.622	0.646	0.669	0.705	0.714	0.74	0.761	0.789
	2	0.617	0.627	0.669	0.693	0.724	0.727	0.771	0.791	0.842

Table III: Results of various XAI algorithms + FSD for Attack Case I in post-training attack strategy

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.668	0.688	0.712	0.739	0.791	0.843	0.868	0.897	0.942
	1	0.738	0.756	0.769	0.787	0.807	0.856	0.877	0.921	0.967
	2	0.78	0.787	0.801	0.817	0.841	0.872	0.888	0.932	0.982
CAM	0.5	0.665	0.685	0.709	0.737	0.788	0.842	0.865	0.895	0.94
	1	0.735	0.755	0.766	0.783	0.805	0.855	0.875	0.918	0.965
	2	0.779	0.784	0.798	0.815	0.839	0.869	0.884	0.928	0.978
LIME	0.5	0.651	0.671	0.695	0.723	0.774	0.828	0.851	0.881	0.926
	1	0.721	0.741	0.752	0.769	0.791	0.841	0.861	0.904	0.951
	2	0.765	0.77	0.784	0.801	0.825	0.855	0.87	0.914	0.991
FSD	0.5	0.568	0.588	0.612	0.64	0.691	0.745	0.768	0.798	0.843
	1	0.638	0.658	0.669	0.686	0.708	0.758	0.778	0.821	0.868
	2	0.682	0.687	0.701	0.718	0.742	0.772	0.787	0.831	0.908

Table IV: Results of various XAI algorithms + FSD for Attack Case II in post-training attack strategy

The outcome of this testing of different algorithms which were made interchangeable in the pipeline suggest that SHAP algorithm was the most accurate which make sense since SHAP is built to work well with RNN which in turn is typically associated with time-series dataset. Most improtantly, it can be observed that the addition of XAI numerical values provide the classifier with more information to make better distinguisments. The manipulation does seem to affect the XAI representation significantly of the data as the important areas become too dissimilar between the false and real data which in turn make it easier for classifier to work. So once more, the addition of XAI values provide an advantage for the classifiers.

B. XFSD architecture positional attacks

With the architecture settled in terms of scenario and algorithms, we are using similar visualisation of aggregates to review the difference of different positional attacks using XAI.

1) Attack case III, IV, V in pre-training: We have exploited the sequential relationship between the data samples from different time-steps in XFSD so it could be an option for adversaries to use positional attacks to bypass the classifiers. For the positional attacks we are talking about three cases where in case III all legitimate data appears before the maliscious data, IV all malscious samples appear after the maliscious data, V maliscious data appears alternately between legitimate data samples. Dependent on the adversarial rate the model in question may ignoere parts of adversarial samples e.g. with 5 % of adversarial data, the model in question learn 95% of legitimate data before learning from the rest of 5% maliscious data, which have less impact on the final performance in comparison to 55% legitimate and 45% maliscious samples. This impact is also shown in tables V VIVII i.e. the fact that higher rates of adversarial data cause higheer negative impact on the knowledge gained by any model from the legitimate data. While the impact is low the model show the highest F-measure in pre-training scenario for attack case IV across all models however,when the impact is increased the model in context of III perform the best. As throguhout the experimentation the outcomes show similar tendencies across model types with the main difference being the overall performance in term of f-measure where XFSD generally is proven to be better performing due to the data enrichment which is also impacted in correlation to the adversarial strategies. The performance of LIME is somehow the same irregardless since it does not consider the sequential order so the impact of the XAI values from that algorithm to the base FSD is minimum. On certain occasions, we dip beneath random guessing which occurrences in cases where the adversarial is $\leq 45\%$ and the deviation is high which is the most severe cases but this is also applicable for cases of base FSD (no XAI enriching) and LIME. The reasons for this we have discussed earlier, but what is interesting is that enriching the FSD vectors with XAI values can take it above random guessing which is significant and an outcome only found via experimentation.

2) Attack case III, IV, V in Post-training: Similar to the positional attacks for pre-training strategy using different XAI algorithms, we conducted the same experiments for the post-training. The results are reflected in table VIII, IX and XV and as before the comparative tendencies across algorithms are the same effectively showing that XAI algorithms does not differ in impact depending on the order of maliscious samples or the distribution of malicious samples which is to be expected as the impact of that is reflected in the LSTM models whereas XAI is a reflection of the explanations (as numerical values) from said model.

Similar trends as we have observed earlier across various algorithms and with FSD with SHAP RNN being the most accurate whilst showcasing similar effects of the positional attacks as was found during the original FSD experimentation.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.971	0.865	0.83	0.816	0.801	0.78	0.775	0.755	0.69
	1	0.882	0.846	0.804	0.788	0.764	0.755	0.754	0.73	0.645
	2	0.847	0.817	0.78	0.76	0.745	0.737	0.725	0.694	0.611
CAM	0.5	0.901	0.856	0.815	0.806	0.792	0.756	0.74	0.722	0.701
	1	0.856	0.825	0.784	0.764	0.767	0.731	0.696	0.679	0.646
	2	0.809	0.789	0.75	0.723	0.701	0.689	0.667	0.644	0.594
LIME	0.5	0.83	0.785	0.744	0.735	0.721	0.685	0.669	0.651	0.63
	1	0.785	0.754	0.713	0.693	0.696	0.66	0.625	0.608	0.575
	2	0.738	0.718	0.679	0.652	0.63	0.618	0.596	0.573	0.523
FSD	0.5	0.842	0.736	0.701	0.687	0.672	0.651	0.646	0.626	0.561
	1	0.753	0.717	0.675	0.659	0.635	0.626	0.625	0.601	0.516
	2	0.718	0.688	0.651	0.631	0.616	0.608	0.596	0.565	0.482

Table V: F-measures for Attack Case III (benign first) with attack samples generated by Attack Case I in pre-training for different XAI algorithms.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.958	0.894	0.858	0.851	0.838	0.822	0.813	0.787	0.729
	1	0.904	0.858	0.824	0.814	0.797	0.777	0.786	0.756	0.672
	2	0.877	0.822	0.787	0.766	0.76	0.76	0.746	0.734	0.646
CAM	0.5	0.841	0.796	0.755	0.746	0.732	0.696	0.68	0.662	0.641
	1	0.796	0.765	0.724	0.704	0.707	0.671	0.636	0.619	0.586
	2	0.749	0.729	0.69	0.663	0.641	0.629	0.607	0.584	0.534
LIME	0.5	0.77	0.725	0.684	0.675	0.661	0.625	0.609	0.591	0.57
	1	0.725	0.694	0.653	0.633	0.636	0.6	0.565	0.548	0.515
	2	0.678	0.658	0.619	0.592	0.57	0.558	0.536	0.513	0.463
FSD	0.5	0.783	0.719	0.683	0.676	0.663	0.647	0.638	0.612	0.554
	1	0.729	0.683	0.649	0.639	0.622	0.602	0.611	0.581	0.497
	2	0.702	0.647	0.612	0.591	0.585	0.585	0.571	0.559	0.471

Table VI: F-measures for Attack Case IV (malicious first) with attack samples generated by Attack Case I in pre-training.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.954	0.907	0.871	0.82	0.796	0.782	0.773	0.743	0.664
	1	0.93	0.855	0.818	0.795	0.765	0.746	0.739	0.706	0.615
	2	0.889	0.827	0.783	0.761	0.735	0.713	0.683	0.658	0.593
CAM	0.5	0.814	0.769	0.728	0.719	0.705	0.669	0.653	0.635	0.614
	1	0.769	0.738	0.697	0.677	0.68	0.644	0.609	0.592	0.559
	2	0.722	0.702	0.663	0.636	0.614	0.602	0.58	0.557	0.507
LIME	0.5	0.77	0.725	0.684	0.675	0.661	0.625	0.609	0.591	0.57
	1	0.725	0.694	0.653	0.633	0.636	0.6	0.565	0.548	0.515
	2	0.678	0.658	0.619	0.592	0.57	0.558	0.536	0.513	0.463
FSD	0.5	0.782	0.735	0.699	0.648	0.624	0.610	0.601	0.571	0.492
	1	0.758	0.683	0.646	0.623	0.593	0.574	0.567	0.534	0.443
	2	0.717	0.655	0.611	0.589	0.563	0.541	0.511	0.486	0.421

Table VII: F-measures for Attack Case V with attack samples generated by Attack Case I in pre-training.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.698	0.714	0.75	0.78	0.814	0.833	0.88	0.899	0.931
	1	0.731	0.754	0.78	0.812	0.858	0.876	0.909	0.932	0.965
	2	0.777	0.78	0.801	0.833	0.873	0.894	0.931	0.95	0.981
CAM	0.5	0.656	0.669	0.696	0.719	0.743	0.764	0.791	0.822	0.862
	1	0.68	0.699	0.717	0.734	0.756	0.799	0.812	0.864	0.905
	2	0.687	0.708	0.722	0.755	0.775	0.816	0.832	0.899	0.956
LIME	0.5	0.585	0.598	0.625	0.648	0.672	0.693	0.72	0.751	0.791
	1	0.609	0.628	0.646	0.663	0.685	0.728	0.741	0.793	0.834
	2	0.616	0.637	0.651	0.684	0.704	0.745	0.761	0.828	0.885
FSD	0.5	0.599	0.615	0.651	0.681	0.715	0.734	0.781	0.800	0.832
	1	0.632	0.655	0.681	0.713	0.759	0.777	0.810	0.833	0.866
	2	0.678	0.681	0.702	0.734	0.774	0.795	0.832	0.851	0.903

Table VIII: F-measures for Attack Case III (benign first) with attack samples generated by Attack Case I in post-training.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.617	0.666	0.718	0.739	0.767	0.796	0.826	0.873	0.915
	1	0.664	0.727	0.731	0.782	0.835	0.848	0.897	0.899	0.93
	2	0.727	0.737	0.778	0.809	0.852	0.871	0.913	0.929	0.981
CAM	0.5	0.596	0.609	0.636	0.659	0.683	0.704	0.731	0.762	0.802
	1	0.62	0.639	0.657	0.674	0.696	0.739	0.752	0.804	0.845
	2	0.627	0.648	0.662	0.695	0.715	0.756	0.772	0.839	0.896
LIME	0.5	0.525	0.538	0.565	0.588	0.612	0.633	0.66	0.691	0.731
	1	0.549	0.568	0.586	0.603	0.625	0.668	0.681	0.733	0.774
	2	0.556	0.577	0.591	0.624	0.644	0.685	0.701	0.768	0.825
FSD	0.5	0.487	0.536	0.588	0.609	0.637	0.666	0.696	0.743	0.785
	1	0.534	0.597	0.601	0.652	0.705	0.718	0.767	0.769	0.800
	2	0.597	0.607	0.648	0.679	0.722	0.741	0.783	0.799	0.851

Table IX: F-measures for Attack Case IV (malicious first) with samples generated by Attack Case I in post-training.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.699	0.731	0.778	0.803	0.849	0.869	0.9	0.923	0.98
	1	0.712	0.751	0.796	0.824	0.868	0.891	0.934	0.958	0.981
	2	0.726	0.781	0.8	0.852	0.891	0.91	0.959	0.979	0.989
CAM	0.5	0.616	0.629	0.656	0.679	0.703	0.724	0.751	0.782	0.822
	1	0.64	0.659	0.677	0.694	0.716	0.759	0.772	0.824	0.865
	2	0.647	0.668	0.682	0.715	0.735	0.776	0.792	0.859	0.916
LIME	0.5	0.535	0.548	0.575	0.598	0.622	0.643	0.67	0.701	0.741
	1	0.559	0.578	0.596	0.613	0.635	0.678	0.691	0.743	0.784
	2	0.566	0.587	0.601	0.634	0.654	0.695	0.711	0.778	0.835
FSD	0.5	0.579	0.611	0.658	0.683	0.729	0.749	0.780	0.803	0.860
	1	0.592	0.631	0.676	0.704	0.748	0.771	0.814	0.838	0.878
	2	0.606	0.661	0.680	0.732	0.771	0.790	0.839	0.859	0.901

Table X: F-measures for Attack Case V post-training attack strategy with attack samples generated by Attack Case I.

C. Advanced data poisoning

Taking inspiration from the work of Miao et.al. [29] we added cases using more advanced adversarial techniques based on their optimal attack framework which uses a bi-level optimization. The two types of data poisoning attacks i.e. the availability attack and the target attack, was applied to our work to evaluate our models effectiveness with the addition of XAI values.

The outcomes which is mainly on the pre-training strategy (data poisoning) showcase the optimization and these types of attacks are highly efficient, even with the addition of XAI values, rendering the system beneath random guessing during some adversarial configurations.

The order in which the results are presented are **1) SHAP, 2) CAM, 3)LIME,4) FSD**.

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
Optimal	0.701	0.681	0.642	0.615	0.593	0.581	0.559	0.536	0.486
Optimal	0.686	0.666	0.627	0.6	0.578	0.566	0.544	0.521	0.471
Optimal	0.674	0.654	0.615	0.588	0.566	0.554	0.532	0.509	0.459
Optimal	0.661	0.641	0.602	0.575	0.553	0.541	0.519	0.496	0.446

Table XI: Optimized Data poisoning availability attack case VI for the scenario of μ

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
Optimal	0.687	0.677	0.674	0.667	0.63	0.616	0.574	0.525	0.477
Optimal	0.674	0.664	0.661	0.654	0.617	0.603	0.561	0.512	0.464
Optimal	0.662	0.652	0.649	0.642	0.605	0.591	0.549	0.5	0.452
Optimal	0.647	0.637	0.634	0.627	0.59	0.576	0.534	0.485	0.437

Table XII: Optimized Data poisoning availability attack case VI for the scenario with σ

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
Optimal	0.788	0.757	0.716	0.696	0.699	0.663	0.628	0.611	0.578
Optimal	0.638	0.607	0.566	0.546	0.549	0.513	0.478	0.461	0.428
Optimal	0.625	0.594	0.553	0.533	0.536	0.5	0.465	0.448	0.415
Optimal	0.613	0.582	0.541	0.521	0.524	0.488	0.453	0.436	0.403

Table XIII: Optimized Data poisoning target attack case VII for the scenario with μ

λ	Adversarial Rate								
	5%	10%	15%	20%	25%	30%	35%	40%	45%
Optimal	0.717	0.707	0.704	0.697	0.66	0.646	0.604	0.555	0.507
Optimal	0.702	0.692	0.689	0.682	0.645	0.631	0.589	0.54	0.492
Optimal	0.677	0.667	0.664	0.657	0.62	0.606	0.564	0.515	0.467
Optimal	0.662	0.652	0.649	0.642	0.605	0.591	0.549	0.5	0.452

Table XIV: Optimized Data poisoning target attack case VII for the scenario of σ

Measured in f-measure we repeated the experimentation for an attack scenario using optimized attack strategy approach. The outcome for this experimentation for our standard deviation and adversarial rate approach along with the newly introduced optimized strategy denoted results that can give us further insight into the robustness of our model. Our model is still performing well within the area but in this attack case scenario it dips beneath random in certain configurations, especially, configurations using the optimized attack strategy approach. The general consensus of XAI enrichment being an advantage is still observable. It is noted that the advanced attack strategies with the optimization is more effective in skewing the perception of the classifiers.

D. Aggregation Method

To possibly reduce some of the dimensionality while retaining representitive vectors we can apply an aggregation for XAI values. In its simplest form this is a summation or mean (or weighted average) that result in a scalar value which is suggested across multiple papers in context of XAI algorithms mostly in multivariate cases. More advanced aggregation methods could be using ML methods such as bagging or time-window aggregation. Another way to get representative values is to extract the distribution(s) and aggregate those values to attain a representative value which is used in multiple works.

	λ	Adversarial Rate								
		5%	10%	15%	20%	25%	30%	35%	40%	45%
SHAP	0.5	0.865	0.813	0.825	0.817	0.799	0.765	0.749	0.732	0.71
	1	0.836	0.766	0.793	0.773	0.779	0.74	0.707	0.688	0.655
	2	0.798	0.721	0.76	0.733	0.712	0.703	0.676	0.653	0.603
CAM	0.5	0.77	0.725	0.684	0.675	0.661	0.625	0.609	0.591	0.57
	1	0.725	0.694	0.653	0.633	0.636	0.6	0.565	0.548	0.515
	2	0.678	0.658	0.619	0.592	0.57	0.558	0.536	0.513	0.463
LIME	0.5	0.734	0.689	0.648	0.639	0.625	0.589	0.573	0.555	0.534
	1	0.689	0.658	0.617	0.597	0.6	0.564	0.529	0.512	0.479
	2	0.642	0.622	0.583	0.556	0.534	0.522	0.5	0.477	0.427
FSD	0.5	0.724	0.681	0.64	0.631	0.617	0.579	0.566	0.547	0.526
	1	0.681	0.649	0.609	0.587	0.591	0.556	0.521	0.503	0.471
	2	0.633	0.613	0.575	0.548	0.525	0.513	0.492	0.467	0.417

Table XV: F-measures for Attack Case I pre-training attack strategy where the default aggregation (simple summation) is applied before concatenation of fixed length vectors (from time-window)

VIII. Discussion

The purpose of XFSD was to improve upon the existing FSD framework. The landscape of MCS for IoT present a plethora of research challenges with one of the more prominent ones being addressing data trustworthiness. With FSD we attempted to build a holistic framework or take a first step toward that and with XFSD we attempt to find a way of improving the robustness of the models that can be derived from the framework by leveraging advanced deep learning capabilities and explainable enrichment.

Fundamentally, the regular FSD rely on deviations between expected (predicted) and real (received) values denoted by the DL models where the received value may be malicious. The addition XAI into XFSD adds more knowledge of the DL model when gaining the predicted value. XAI assign values to each feature in the sequence resulting in different patterns which can make it distinguishable when classifying as adversarial attacks also affect the XAI values enabling comparisons of XAI values from trained benign models and adversarial models from testing. To aspects of note is that XAI values are also affected by adversarial strategies. With post-training we train a model on benign samples and compute XAI values which can be considered benign which are then used for comparisons to malicious samples. In pre-traiing, the accuracy is dependent on the adversarial rate in the process which dependent on how many adversarial samples is present determines the validity of the XAI values i.e. if the dataset in the time-frame is significantly predominately benign the XAI values is more significant resulting in better accuracy and vice versa if the the dominance switches to malicious the accuracy is negatively affected significantly. Essentially, with post-training we have a expected pattern from the XAI values which can be compared with the real (received) values in establishing a class/boundary and with the data poisoning the impact is reflected in how skewed the perception of the model is which we can observe is determined mainly by the adversarial rate which makes sense since the discrepancy of benign XAI values and affected XAI values should not be significant while the distribution is still significantly benign. The breaking point for the significance is observed to be around which is related as spikes in terms of performance. Furthermore, indication show that all models converges to random guessing. However, SHAP seem to handle this convergence which in that scenario happens at an adversarial rate of 20%.

For the evaluation we used a vector consistent of difference distance measures comprising data vectors of all 5 features in a given time-frame originating from the data sources of a variety of trustworthiness levels containing benign and malicious samples concatenated with corresponding explainable values in the form of quantifiable numerical values. The segmentation of trustworthiness levels was denoted by a factor applied to standard deviation of each feature ensuring that the data distribution considered for each feature was balanced. The challenges that comes with mainly pertains to differing behavior over time for each feature and also unforeseen changes typically called concept drift.

Also, with colluding adversaries attacking the data collection process by feeding classifiers with falsified data, it is evident that not does the order matter but so does the time-window. In this project we experimented with different to find the ideal window. Furthermore, we have tried different algorithms both for XAI and DL. In the future this can be done more exhaustively but we have attempted to cover most general ground and provide a framework which incorporates explainable values.

With this sort of framework it is always possible to add more i.e. add more algorithms, attack strategies, aggregation methods an so forth. Future, works can expand upon this general framework and aim toward a more exhaustive experimentation.

IX. Conclusion

The general conclusion that could be reached was that the enrichment of XAI values provided an observable improvement in terms of classifiers being able to distinguish between legit and malicious samples. We also showcased that for time-series data using LSTM/RNN in combination with SHAP was more accurate than applying it with CAM which can be explained by the nature in which CAM works in combination with CNN for time-series i.e. CAM considers areas of importance whereas SHAP considers the whole ordinal information sequence. This is observed to be advantagoues in dataset such as ours which has to consider issue pertaining to concept drift i.e. unforeseen changes over time. The information that can be extracted from sequential order is further proven important by the improvements in accuracy for the classifiers that can be observed when introducing XAI values and its comparative better performance when comparing it to non-sequential algorithms such as LIME or the base model which has no enrichment with XAI values.

X. Acknowledgment

This work was supported by Innovation Fund Denmark (IFD) under SecDNS project and the European Commission under the STAR project, Grant Agreements no. 956573.

References

[1] T. Rojat, R. Puget, D. Filliat, J. Del Ser, R. Gelin, and N. Diaz Rodriguez, "Explainable artificial intelligence (xai) on timeseries data: A survey," 04 2021.

[2] S. Gisdakis, T. Giannetsos, and P. Papadimitratos, "Shield: a data verification framework for participatory sensing systems," 07 2015.

[3] N. Banerjee, T. Giannetsos, E. Panaousis, and C. Cheong Took, "Unsupervised learning for trustworthy iot," 07 2018, pp. 1–8.

[4] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, Nov. 2019.

[5] S. Afzal-Houshmand, S. Homayoun, and T. Giannetsos, "A perfect match: Deep learning towards enhanced data trustworthiness in crowdsensing systems," in *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, 2021, pp. 258–264.

[6] B. Guo, Z. Yu, X. Zhou, and D. Zhang, "From participatory sensing to mobile crowd sensing," in *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, 2014, pp. 593–598.

[7] X. Li, K. Xie, X. Wang, G. Xie, D. Xie, Z. Li, J. Wen, Z. Diao, and T. Wang, "Quick and accurate false data detection in mobile crowd sensing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1339–1352, 2020.

[8] L. Cheng, L. Kong, C. Luo, J. Niu, Y. Gu, W. He, and S. Das, "Deco: False data detection and correction framework for participatory sensing," in *2015 IEEE 23rd International Symposium on Quality of Service (IWQoS)*. IEEE, Jun. 2015.

[9] K. E. Mokhtari, B. P. Higdon, and A. Başar, "Interpreting financial time series with shap values," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '19. USA: IBM Corp., 2019, p. 166–172.

[10] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, p. 18, 12 2020.

[11] D. Dataman. Explain your model with the shap values. [Online]. Available: https://towardsdatascience.com/explain-your-model-with-the-shap-values-bc36aac4de3d

[12] B. Khaleghi. The how of explainable ai: Post-modelling explainability. [Online]. Available: https://towardsdatascience.com/the-how-of-explainable-ai-post-modelling-explainability-8b4cbc7adf5f

[13] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1578–1585.

[14] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Accurate and interpretable evaluation of surgical skills from kinematic data using fully convolutional neural networks," 08 2019.

[15] F. Oviedo, Z. Ren, S. Sun, C. M. Settens, Z. Liu, N. T. P. Hartono, R. Savitha, B. L. DeCost, S. I. P. Tian, G. Romano, A. G. Kusne, and T. Buonassisi, "Fast classification of small x-ray diffraction datasets using data augmentation and deep neural networks," *ArXiv*, vol. abs/1811.08425, 2018.

[16] K. Kashiparekh, J. Narwariya, P. Malhotra, L. Vig, and G. Shroff, "Convtimenet: A pre-trained deep convolutional neural network for time series classification," 07 2019, pp. 1–8.

[17] L. Zhou, C. Ma, X. Shi, D. Zhang, W. Li, and L. Wu, "Salience-cam: Visual explanations from convolutional neural networks via salience score," 07 2021, pp. 1–8.

[18] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144. [Online]. Available: https://doi.org/10.1145/2939672.2939778

[19] L. Hulstaert. Understanding model predictions with lime. [Online]. Available: https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b

[20] A. Sharma. Decrypting your machine learning model using lime. [Online]. Available: https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5

[21] I. Kakogeorgiou and K. Karantzalos, "Evaluating explainable artificial intelligence methods for multi-label deep learning classification tasks in remote sensing," 04 2021.

[22] R. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[23] C. Miao, Q. Li, H. Xiao, W. Jiang, M. Huai, and L. Su, "Towards data poisoning attacks in crowd sensing systems," 06 2018, pp. 111–120.

[24] D. Mercier, A. Dengel, and S. Ahmed, "Patchx: Explaining deep models by intelligible pattern patches for time-series classification," 02 2021.

[25] J.-Y. Kim and S.-B. Cho, "Electric energy consumption prediction by deep learning with state explainable autoencoder," *Energies*, vol. 12, p. 739, 02 2019.

[26] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *ArXiv*, vol. abs/1705.07874, 2017.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.

[28] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2921–2929.

[29] C. Miao, Q. Li, H. Xiao, W. Jiang, M. Huai, and L. Su, "Towards data poisoning attacks in crowd sensing systems," 06 2018, pp. 111–120.

CHAPTER 5

Conclusive Outlook from Outcomes

This PhD study general objective was to conduct an investigation into Adversarial Machine Learning in Cybersecurit with a disposition starting from the SecDNS project using DNS data to create machine learning frameworks towards cyber crime prevention. Throughout, this expanded to also include other data sources such as mobile crowd sourcing.

We have been able to establish the major advantages that using Machine learning methods provides in the realm of cybersecurity using Data scientific approach. As a results, we have been able to create frameworks and experimental approaches that can be applied to multitude of different contexts with similar objectives.

The scope of artificial intelligence is quite broad, even if we try to restrict it to machine learning or deep learning for that matter. The hope for the future is that the area get further investigated in a great variety of contexts and with a great variety in data sources.

Bibliography

[1] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*, BICT'15, p. 21–26, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[2] A. A. Diro and N. K. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Comp. Syst.*, vol. 82, pp. 761–768, 2017.

[3] B. L. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pp. 581–585, 2016.

[4] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring: A survey," 2016.

[5] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[6] A. Tuor, R. Baerwolf, N. Knowles, B. Hutchinson, N. Nichols, and R. Jasper, "Recurrent neural network language models for open vocabulary event-level cyber anomaly detection," 2017.

[7] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," 2017.

[8] A. Brown, A. Tuor, B. Hutchinson, and N. Nichols, "Recurrent neural network attention mechanisms for interpretable system log anomaly detection," in *Proceedings of the First Workshop on Machine Learning for Computing Systems*, MLCS'18, (New York, NY, USA), Association for Computing Machinery, 2018.

[9] B. J. Radford, B. D. Richardson, and S. E. Davis, "Sequence aggregation rules for anomaly detection in computer network traffic," 2018.

[10] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, Aug. 2019.

[11] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," 2015.

[12] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," 2018.

[13] N. Banerjee, T. Giannetsos, E. Panaousis, and C. C. Took, "Unsupervised learning for trustworthy iot," *CoRR*, vol. abs/1805.10401, 2018.

[14] N. Baracaldo, B. Chen, H. Ludwig, A. Safavi, and R. Zhang, "Detecting poisoning attacks on machine learning in iot environments," in *2018 IEEE International Congress on Internet of Things (ICIOT)*, pp. 57–64, July 2018.

[15] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," *CoRR*, vol. abs/1710.00942, 2017.

[16] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, ACM, 2011.

[17] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *arXiv preprint arXiv:1511.03034*, 2015.

[18] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 16–25, ACM, 2006.

[19] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, *et al.*, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 99–108, ACM, 2004.

[20] S. Gisdakis, T. Giannetsos, and P. Papadimitratos, "SHIELD: A data verification framework for participatory sensing systems," in *8th ACM Conf. on Security & Privacy in Wireless and Mobile Networks*, p. 16, 2015.

[21] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[22] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[23] Y. Vorobeychik, "Adversarial ai.," in *IJCAI*, pp. 4094–4099, 2016.

[24] M. Barreno, B. Nelson, A. D. Joseph, and J. Tygar, "The security of machine learning," *Machine Learning*, vol. 81, no. 2, pp. 121–148, 2010.

[25] S. Mokhtarani, "Embeddings in machine learning: Everything you need to know." https://www.featureform.com/post/the-definitive-guide-to-embeddings.

[26] pascal brokmeier, "An overview of categorical input handling for neural networks."

[27] "Starspace." https://github.com/facebookresearch/StarSpace.

[28] P. Shreyas, "Deep embedding's for categorical variables (cat2vec)." https://towardsdatascience.com/deep-embeddings-for-categorical-variables-cat2vec-b05c8ab63ac0", addendum =.

[29] E. Ross, "Building categorical embeddings." https://skeptric.com/categorical-embeddings/", addendum =.

[30] N. Takama and D. P. Loucks, "Multi-level optimization for multi-objective problems," *Applied Mathematical Modelling*, vol. 5, no. 3, pp. 173–178, 1981.

[31] R. Achddou, J. M. Di Martino, and G. Sapiro, "Nested learning for multi-level classification," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2815–2819, 2021.

[32] J. Li, L. Wu, R. Guo, C. Liu, and H. Liu, "Multi-level network embedding with boosted low-rank matrix approximation," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '19, (New York, NY, USA), p. 49–56, Association for Computing Machinery, 2019.

[33] M. Köppel, A. Segner, M. Wagener, L. Pensel, A. Karwath, and S. Kramer, *Pairwise Learning to Rank by Neural Networks Revisited: Reconstruction, Theoretical Analysis and Practical Performance*, pp. 237–252. 04 2020.

[34] "Stellargraph." https://stellargraph.readthedocs.io/en/stable/demos/index.html.

[35] W. Yu, N. Lu, X. Qi, P. Gong, and R. Xiao, "Pick: Processing key information extraction from documents using improved graph learning-convolutional networks," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 4363–4370, 2021.

[36] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, "Beyond one-hot encoding: Lower dimensional target embedding," *Image and Vision Computing*, vol. 75, pp. 21–31, 2018.

[37] O. Ben-Ayed, "Bilevel linear programming," *Computers Operations Research*, vol. 20, no. 5, pp. 485–501, 1993.

[38] C. Mougan, D. Masip, J. Nin, and O. Pujol, "Quantile encoder: Tackling high cardinality categorical features in regression problems," 05 2021.

[39] R. Achddou, J. Martino, and G. Sapiro, "Nested learning for multi-granular tasks," 07 2020.

[40] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, (New York, NY, USA), 2016.

[41] S. Palachy, "Document embedding techniques." `https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d`.

[42] P. Godec, "Graph embeddings — the summary)." `https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007"`, addendum =.

[43] R. Y. R. S. Jure Leskovec, William L. Hamilton, "Representation learning on networks."

[44] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, 05 2014.

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] C. Nicholson, "A beginner's guide to neural networks and deep learning," 2019.

[47] T. Lochmann and S. Deneve, "Neural processing as causal inference," *Current opinion in neurobiology*, vol. 21, pp. 774–81, 07 2011.

[48] L.-f. REN, Z. WANG, X. LIU, Q.-s. LI, and Z. Chen, "The design and implementation of a covering mdn-complete-life-cycle malicious domain detection framework," *DEStech Transactions on Computer Science and Engineering*, 07 2017.

[49] T. Gupta, "Deep learning: Feedforward neural network," 2017.

[50] A. Shekhovtsov, B. Flach, and M. Busta, "Feed-forward uncertainty propagation in belief and neural networks," 03 2018.

[51] K. Yoon, R. Liao, Y. Xiong, L. Zhang, E. Fetaya, R. Urtasun, R. Zemel, and X. Pitkow, "Inference in probabilistic graphical models by graph neural networks," 03 2018.

[52] P. Lison and V. Mavroeidis, "Automatic detection of malware-generated domains with recurrent neural models," 09 2017.

[53] V. R, S. Kp, and P. Poornachandran, "Detecting malicious domain names using deep learning approaches at scale," *Journal of Intelligent and Fuzzy Systems*, vol. 34, pp. 1355–1367, 03 2018.

[54] T. Rojat, R. Puget, D. Filliat, J. Del Ser, R. Gelin, and N. Diaz Rodriguez, "Explainable artificial intelligence (xai) on timeseries data: A survey," 04 2021.

[55] D. Mercier, A. Dengel, and S. Ahmed, "Patchx: Explaining deep models by intelligible pattern patches for time-series classification," 02 2021.

[56] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), p. 1135–1144, Association for Computing Machinery, 2016.

[57] K. E. Mokhtari, B. P. Higdon, and A. Başar, "Interpreting financial time series with shap values," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, CASCON '19, (USA), p. 166–172, IBM Corp., 2019.

[58] J.-Y. Kim and S.-B. Cho, "Electric energy consumption prediction by deep learning with state explainable autoencoder," *Energies*, vol. 12, p. 739, 02 2019.

[59] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *ArXiv*, vol. abs/1705.07874, 2017.

[60] L. Zhou, C. Ma, X. Shi, D. Zhang, W. Li, and L. Wu, "Salience-cam: Visual explanations from convolutional neural networks via salience score," pp. 1–8, 07 2021.

[61] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.

[62] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1578–1585, 2017.

[63] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Accurate and interpretable evaluation of surgical skills from kinematic data using fully convolutional neural networks," 08 2019.

[64] F. Oviedo, Z. Ren, S. Sun, C. M. Settens, Z. Liu, N. T. P. Hartono, R. Savitha, B. L. DeCost, S. I. P. Tian, G. Romano, A. G. Kusne, and T. Buonassisi, "Fast classification of small x-ray diffraction datasets using data augmentation and deep neural networks," *ArXiv*, vol. abs/1811.08425, 2018.

[65] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929, 2016.

[66] R. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: current state and future challenges," *IEEE Communications Magazine*, vol. 49, pp. 32–39, Nov. 2011.

[67] N. Pitropakis, E. Panaousis, T. Giannetsos, E. Anastasiadis, and G. Loukas, "A taxonomy and survey of attacks against machine learning," *Computer Science Review*, vol. 34, p. 100199, Nov. 2019.

[68] E. Cinlar, *Probability and Stochastics - 2011.* Springer, 2011.

[69] F. Dekking, C. Kraaikamp, H. Lopuhaä, and L. Meester, *A modern introduction to probability and statistics. Understanding why and how.* 01 2005.

[70] J. Brownlee, "A gentle introduction to joint, marginal, and conditional probability," 2019.

[71] R. J. Trumpler and H. F. Weaver, "Statistical astronomy," 1962.

[72] A. Everitt, B. S.; Skrondal, "Cambridge dictionary of statistics," 2010.

[73] J. H. Joseph K. Blitzstein, "Introduction to probability," 2014.

[74] R. Durrett, "Probability. theory and examples," 01 2010.

[75] P. A. Gagniuc, "Markov chains: From theory to implementation and experimentation, first edition," 2017.

[76] J. Kim, Jin H.; Pearl, "A computational model for causal and diagnostic reasoning in inference systems," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983.

[77] P. Wenig, "Belief propagation in bayesian networks," 2019.

[78] I. Khalil, T. Yu, and B. Guan, "Discovering malicious domains through passive dns data graph analysis," 06 2016.

[79] I. Khalil, B. Guan, M. Nabeel, and T. Yu, "Killing two birds with one stone: Malicious domain detection with high accuracy and coverage," 11 2017.

[80] D. M. . D. Rodriguez, "Beyond lexical and pdns: using signals on graphs to uncover online threats at scale," *Cisco Umbrella (OpenDNS), USA*, 2017.

[81] L. Chang, Z.-J. Zhou, Y. You, L.-H. Yang, and Z.-G. Zhou, "Belief rule based expert system for classification problems with new rule activation and weight calculation procedures," *Information Sciences*, vol. 336, 12 2015.

[82] H. Tran, A. Nguyen, P. Vo, and T. Vu, "Dns graph mining for malicious domain detection," pp. 4680–4685, Dec 2017.

[83] J. Yedidia, W. Freeman, and Y. Weiss, *Understanding belief propagation and its generalizations*, vol. 8, pp. 239–269. 01 2003.

[84] J. Diebel and S. Thrun, "An application of markov random fields to range sensing.," *Advances in Neural Information Processing Systems*, vol. 5, 01 2005.

[85] M. Ibrahimi, A. Javanmard, Y. Kanoria, and A. Montanari, "Robust max-product belief propagation," *Circuits, Systems and Computers, 1977. Conference Record. 1977 11th Asilomar Conference on*, 11 2011.

[86] P. Manadhata, S. Yadav, P. Rao, and W. Horne, "Detecting malicious domains via graph inference," pp. 1–18, 09 2014.

[87] A. Nath and P. Domingos, "Approximate lifted belief propagation," *AAAI Workshop - Technical Report*, 11 2010.

[88] H. Tran, C. Dang, H. Nguyen, P. Vo, and T. Vu, "Multi-confirmations and dns graph mining for malicious domain detection," 2019.

[89] W. Link and R. Barker, "Bayesian inference," *Bayesian Inference*, 01 2010.

[90] J. Brooks-Bartlett, "Probability concepts explained: Bayesian inference for parameter estimation," 2018.

[91] J. R. Chung and G. Yi, "Belief propagation in bayesian network," 2014.

[92] F. Altarelli, A. Braunstein, L. Dall'Asta, A. Lage-Castellanos, and R. Zecchina, "Bayesian inference of epidemics on networks via belief propagation," *Physical review letters*, vol. 112, p. 118701, 03 2014.

[93] E. Stalmans and B. Irwin, "A framework for dns based detection and mitigation of malware infections on a network," pp. 1–8, Aug 2011.

[94] V. Létal, T. Pevnỳ, V. Šmıdl, and P. Somol, "Finding new malicious domains using variational bayes on large-scale computer network data," pp. 1–10, 2015.

[95] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum, "Stereo matching using belief propagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 787–800, July 2003.

[96] E. Sudderth, A. Ihler, M. Isard, W. Freeman, and A. Willsky, "Nonparametric belief propagation," *Commun. ACM*, vol. 53, pp. 95–103, 10 2010.

{2.7182818284}ϑφερτυθιοπσδφγηξκλ