Frank Beer

A Hybrid Flow-based Intrusion Detection System Incorporating Uncertainty

kassel university press

Frank Beer

A Hybrid Flow-based Intrusion Detection System Incorporating Uncertainty

kassel university press

This work has been accepted by the Faculty of Electrical Engineering and Computer Science of the University of Kassel as a thesis for acquiring the academic degree of Doktor der Naturwissenschaften (Dr. rer. nat.).

Supervisor: Prof. Dr. Bernhard Sick, University of Kassel
Co-Supervisor: Prof. Dr. Ulrich Bühler, Fulda University of Applied Sciences
Co-Supervisor: Prof. Dr. Arno Wacker, Bundeswehr University Munich

Defense day: 21. February 2022

Bibliographic information published by Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at http://dnb.dnb.de.

Zugl.: Kassel, Univ., Diss. 2022
ISBN 978-3-7376-1059-9
DOI: https://doi.org/10.17170/kobra-202207146472

Printing Shop: Print Management Logistik Service, Kassel
Printed in Germany

Abstract

The advances of today's cyberattacks threatening network infrastructures are both versatile and alarming. This requires thoroughly planned security solutions to spot malicious behavior in those networks. Systems serving this duty are intrusion detectors commonly relying on deep packet inspection, which come up with high resource consumption because network traffic is observed at a very fine granularity. With increasing link speeds of current and future networks, this situation is becoming a serious affair for operational staff. These circumstances are further fueled by the rise of end-to-end encryption preventing deeper insights to packet content. To absorb these drawbacks, we investigate alternative roads and propose a new hybrid flow-based intrusion detection system in this work. It rests upon flow data as primary entity to monitor network sites, which is enabled by the established flow export protocols NetFlow/IPFIX. As opposed to packet data, flows elevate network activities to a much coarser format posing several practical benefits. Yet, it is unclear to which degree flows can contribute to a broad attack coverage with a low false alarm rate realized through a single detection system. On this account, a feature analysis is conducted on newly compiled benchmark data to expose meaningful flow features coupled with other supplemental information that are incorporated into our intrusion detector. Moreover, the system adapts the essential idea of combining misuse and anomaly detection techniques based on machine learning principles towards a hybrid solution following a two-step inspection attempt. In the first step, the stream of incoming flows is examined against a repository of known patterns. If no pattern match can be identified at this point, flows are directed to the anomaly detector for a final examination. From there, missing knowledge in the pattern repository is complemented gradually by a new pattern building mechanism employing in-database analytics, i.e. an undertaking to lift database systems beyond traditional data management tasks. A key asset of this cascading design is transparency as black box classifications at the anomaly detector are immediately turned into human readable patterns serving follow-up actions for responsible personnel. Additionally, our system architecture aims at scalability and adaptivity to address network dynamics. Empirical assessments under very realistic circumstances reveal interesting insights. In particular, they confirm that the proposed solution can compensate increasing workloads by appending more hardware resources permitting to monitor medium to large production networks. It can also handle simple concept drift scenarios self-sufficiently but minor manual intervention is required for more rigorous drifts. Furthermore, results document a baseline protection against several attack types. This outcome is paired with few false alarms and a high chance for explainable predictions. These and further findings demonstrate that our approach is a step in the right direction to safeguard network systems without cumbersome packet analysis leaving ample room for further research.

Zusammenfassung

Die Bedrohungslage heutiger Cyberangriffe für Netzinfrastrukturen ist facettenreich und alarmierend. Dies erfordert sorgfältig geplante Sicherheitslösungen, um bösartiges Verhalten in diesen Netzen zu erkennen. Systeme, die sich dieser Aufgabe stellen, sind so genannte Angriffserkennungssysteme. Sie beruhen üblicherweise auf Deep-Packet-Inspection und weisen einen hohen Ressourcenbedarf auf, da der Netzwerkverkehr sehr feingranular aufgezeichnet wird. Mit zunehmender Verbindungsgeschwindigkeit aktueller und zukünftiger Netze wird diese Situation für Betreiber eine immer größere Herausforderung. Diese Umstände werden durch die zunehmende Ende-zu-Ende-Verschlüsselung zusätzlich verschärft, die tiefere Einblicke in den Inhalt der Pakete verhindert. Um diesen Nachteilen entgegenzuwirken, untersuchen wir alternative Wege und schlagen in dieser Arbeit ein neues hybrides, flow-basiertes System zur Erkennung von Angriffen vor. Es stützt sich auf Flow-Daten als primäre Entität zur Überwachung von Netzwerkstandorten, was durch die etablierten Protokolle NetFlow/IPFIX ermöglicht wird. Im Gegensatz zu Paketdaten werden die Netzwerkaktivitäten bei Flows in einem viel gröberen Format dargestellt, was mehrere praktische Vorteile mit sich bringt. Es ist jedoch unklar, inwieweit Flows zu einer breiten Angriffsabdeckung mit einer niedrigen Fehlalarmrate durch ein einziges Erkennungssystem beitragen können. Aus diesem Grund wird eine Merkmalsanalyse auf neuen Benchmark-Daten durchgeführt, um aussagekräftige Flow-Features in Verbindung mit anderen ergänzenden Informationen zu extrahieren, die in unser Erkennungssystem einfließen. Darüber hinaus adaptiert das System die grundlegende Idee der Misuse- und Anomalie-Erkennung auf der Grundlage des maschinellen Lernens hin zu einer hybriden Lösung, die auf einem zweistufigen Inspektionsverfahren fußt. Im ersten Schritt wird der Strom eingehender Flows anhand bekannter Muster in einer Datenbank untersucht. Wenn zu diesem Zeitpunkt keine Übereinstimmung zu einem Muster festgestellt werden kann, werden Flows für eine abschließende Prüfung an die Anomalie-Erkennung weitergeleitet. Von dort aus wird das fehlende Wissen in der Musterdatenbank schrittweise durch eine neue Methode zur Mustererstellung ergänzt. Ein Hauptvorteil dieses kaskadierten Designs ist Transparenz, da Black-Box-Klassifizierungen an der Anomalie-Erkennung sofort in menschenlesbare Muster umgewandelt werden, die dem verantwortlichen Personal für Folgemaßnahmen zur Verfügung stehen. Des Weiteren bietet unsere Systemarchitektur Skalierbarkeit und Adaptivität, um Netzwerkdynamiken zu adressieren. Empirische Bewertungen unter sehr realistischen Bedingungen zeigen interessante Ergebnisse. Sie bestätigen insbesondere, dass die vorgeschlagene Lösung steigende Lasten durch den Einsatz zusätzlicher Hardware-Ressourcen kompensieren kann, was die Überwachung mittlerer bis großer Produktionsnetzwerke ermöglicht. Das System kann ebenfalls einfache Concept-Drifts selbständig behandeln, während extreme Drift-Szenarien ein geringfügiges manuelles

Eingreifen erfordern. Darüber hinaus dokumentieren die Ergebnisse einen gewissen Grundschutz gegenüber verschiedenen Angriffstypen. Dieses Resultat ist gepaart mit wenigen Fehlalarmen und einer hohen Wahrscheinlichkeit für erklärbare Vorraussagen. Diese und weitere Ergebnisse zeigen, dass unser Ansatz ein Schritt in die richtige Richtung ist, um Netzwerksysteme ohne umständliche Paketanalyse zu schützen und viel Raum für weitere Forschung bleibt.

Danksagung

Für den Erfolg einer solchen Arbeit müssen nicht nur die organisatorischen und technischen Rahmenbedingungen gegeben sein. Auch die menschliche Komponente ist von immenser Bedeutung. Viele Menschen haben mich bei der Anfertigung dieser Arbeit sowohl fachlich wie auch persönlich gestützt. Ihnen möchte ich herzlich danken.

Beginnen möchte ich mit Prof. Dr. Bernhard Sick. Zusammen mit seinem Team hat er die nötige Motivation für die Erstellung eines neuen Benchmark-Datensatzes gegeben, welcher wertbringend umgesetzt wurden. Ebenfalls möchte ich ihm für die unkomplizierte organisatorische Unterstützung vor allem in der Schlussphase dieser Arbeit danken. Prof. Dr. Arno Wacker möchte ich für die gute Betreuung über weite Strecken dieser Arbeit danken. In regelmäßigen Terminen wurde zusammen mit seinem Team über Zwischenergebnisse rege diskutiert. Durch diese Impulse wurden viele Aspekte dieser Arbeit mit der nötigen Tiefe geschärft. Ein besonderer Dank gilt Prof. Dr. Ulrich Bühler. Er übernahm das Mentoring über die gesamte Laufzeit der Arbeit und hatte immer ein offenes Ohr für auftretende Probleme und Herausforderungen für die stets eine Lösung gefunden werden konnte. Es entstanden endlose und fruchtbare Diskussionen an die ich mich noch in vielen Jahren erinnern werde. Sie haben mich fachlich wie persönlich maßgeblich geprägt. Darüber hinaus möchte ich ihm für den Freiraum bei der Bearbeitung danken. Sicherlich hätte ohne diese Großzügigkeit die Arbeit nicht in dieser Ausführlichkeit angefertig werden können.

Für die gute Zusammenarbeit und technische Unterstützung möchte ich mich bei EDAG Engineering GmbH und Prof. Dr. Sebastian Rieger bedanken. Dadurch konnte zum einen realistischer Netzwerkverkehr aufgezeichnet werden, welcher die Grundlage für viele Auswertungen in dieser Arbeit darstellt. Zum anderen konnten diese Analysen in einer robusten Infrastruktur reproduzierbar ausgeführt werden, was im Hochschulbetrieb nicht selbstverständlich ist. In diesem Zusammenhang möchte ich weiteren Arbeitskollegen und Studenten danken, die mich auf dieser Reise begleitet haben. Hervorzuheben sind dabei Katharina, David und Tim. Während Katharina tatkräftig bei Implementierungsarbeiten mitwirkte, halfen mir David und Tim bei der praktischen Aufzeichnung von realistischen Angriffsszenarien für den Benchmark-Datensatz “NDSec-1”.

Meinen tiefsten Dank möchte ich meiner Familie und Freunden ausdrücken. Insbesondere sind meine Eltern und mein Bruder zu nennen. Ihr habt den Weg zu dieser Arbeit geebnet. Somit wäre ohne euch all dies nicht möglich gewesen. Die letzten Worte möchte ich an meine Verlobte Eva richten: Du standes immer bedingungslos hinter mir auch in den schwierigsten Zeiten und hast viel Last von mir genommen. Eva, du warst und bleibst immer eine Inspiration für mich.

- Dedicated to my small family. Eva and Luis, this is for you. -

Contents

Part I

Prologue

Chapter 1

Introduction

Securing network systems is increasingly becoming a challenge. This affects both manufacturers building solutions for this duty and operational staff running these solutions to monitor and protect their environments from data theft, resource misuse, outages and other potential damages caused by invaders. This situation has different origins but does not stop at this stage. Even the research community has suffered from some factors impeding significant progress for years. This work attempts to target several issues related thereto by proposing a new system to analyze network traffic continuously and to uncover intrusive behavior. For this purpose, a general introduction is provided in this chapter. First, we contextualize the underlying problem domain our research effort aims at. Second, the contribution is stated. Lastly, an outline is given about the organization of this thesis.

1.1 Problem Description

Despite a high estimated number of unreported data breaches, the trend of cybercrime becomes more apparent and intimidating at the same time. On a monthly basis, we receive headlines of new cyberattacks targeting "information technology" (IT) assets of the public and the private sector and new security vulnerabilities are unveiled weekly, which in turn catalyzes the deployment of new "malicious software" (malware) and its variants every day. From this perspective, it can be easily derived that "intrusion detection systems" (IDSs) are an indispensable key factor to safeguard individual computer systems or entire network infrastructures. In the latter case, intrusion detection typically relies on some sort of deep Internet Protocol (IP) packet analysis because incoming network traffic can be observed accurately based on packet header and payload data respectively. Yet, in-depth data analysis at this fine granularity is very resource intensive. Thus, maintenance tasks of such systems are increasingly becoming a challenge for enterprises and institutions with the growing link speed of current and future networks to consider (e.g. [1, 2, 3]). Another inherent concern of packet-based intrusion detection solutions is the rise of "end-to-end encryption", which prevent deeper insights to the content (e.g [4, 5]). Consequently, it is essential to investigate ways absorbing these drawbacks while supplying equivalent detection capabilities. One encouraging opportunity to spot cyberattacks and other unsolicited activities in networks is "flow monitoring" [6]. In contrast to traditional packet observation, "flows" mainly summarize header information of packets. As such, one gets a coarser view to ongoing network activities serving remarkable advantages in practice. Additionally, the protocol IP Flow Information Export (IPFIX) [7] emerged in the recent past as Internet Standard with dedicated metering and exporting process, which builds upon NetFlow [8]. This way, the integration of flow monitoring functionality into existing network landscapes is facilitated with respect to operational aspects. These developments inspired academia and industry to leverage flow technology to uncover intrusions and, in some way, substantial progress has been made over the last decade and beyond (e.g. [9, 10, 11]). To this day, however, the discipline of "flow-based intrusion detection" can be considered at an early stage and several problems are still to be solved. They require intensive investigation towards the central question whether existing flow technology can be, to some extent, an alternative to conventional packet-based inspection.

Examining IDSs according to their detection capabilities, systems are typically distinguished by their engine which either consists of "misuse-based" or "anomaly-based" detection techniques. While misuse methods attempt to define a model describing intrusive footprints, an anomaly-based approach profiles benign system behaviors alerting deviations from the norm. Due to these complementary approaches, quite natural, both methods convey different characteristic implications in terms of practical aspects. From its definition, misuse detection can precisely identify known attacks with low false alarms but it is incapable to unveil unknown attacks. Conversely, the latter point is the most notable strength of anomaly-based methods, i.e. detecting novel attacks (or zero-day attacks). However, the fundamental drawbacks of anomaly-based IDSs reside in their operational handling: (i) existing anomaly detectors intrinsically generate many false alarms. Considering the sheer amount of traffic to investigate, even a small percentage of incorrect alerts can quickly overwhelm security staff (e.g. [12, 13]) and, thus, under-

mines the system's trustworthiness in the long run. (ii) Certainly, anomaly detection in the classical sense does not depend on prior attack knowledge for a successful detection but it is rather incapable to describe intrusions properly. Therefore, it can be seen as "black box" in the general case preventing insights to the underlying decision-making process. This missing "transparency" is troublesome because no system operator is willing to rely on unexplainable alarms (e.g. [14, 15]). With these arguments at hand, it is not surprising that misuse approaches based on "signatures" are the preferred and predominant choice to protect enterprises and institutions despite the mentioned shortcomings (e.g. [14, 16, 17]). From this last point, it can be deduced that most deployed IDSs indeed provide limited defense for the timely gap between an open "vulnerability" and the availability of attack signatures covering this security leak. Such situations are further fueled by the laborious signature creation process often requiring security experts to manually hand-craft them. As a result, it is of high interest to reduce this gap to prevent potential damage caused by an "exploit" targeting a vulnerability. This demand leads to the automated generation of signatures and, in fact, several sound ideas have been suggested over the years (e.g. [18, 19, 20]). Unfortunately, they mainly concentrate on payload analysis, which is prone to encryption techniques as argued earlier. Another branch of research investigates opportunities to fuse the benefits of misuse and anomaly detectors towards their hybridization. Most representative works either attempt to combine both techniques in a parallel (e.g. [21, 22, 23]) or cascading fashion (e.g. [24, 25, 26]). While the former lacks a certain degree of the aforementioned transparency making them less attractive for operational usage, the latter constellation is very promising especially when incorporating automated signature creation. However, existing systems carry flaws restraining their usage in practice. These limitations include high computational demands restricting "scalability" and the naive consideration of static data distributions, which is a prohibitive assumption in dynamic network environments where data distributions tend to change. This demands appropriate adaption to shifting circumstances. Hence, immediate action is imperative to design and deploy a reliable, efficient and transparent intrusion detection solution to protect current and future network systems.

A final issue penetrating almost all dimensions of "network security research" can be attributed to the limited availability of adequate benchmark data sets, which has been reported in literature repeatedly (e.g. [27, 28, 29]). Even though some data sets are existing, the most frequently used ones among them stem from a particular traffic collection (see [30, 31, 32]) that was devised more than two decades ago. Being aware of the age alone suggests to rely contemporary research on newer network traces reflecting recent requirements regarding traffic and state-of-the-art attack types rather than using those antiquated ones. As a consequence, further attempts have been made to produce more effective data sets but many of those proposed network traces (e.g [33, 34, 35]) pursue specific goals and refuse their application to "general-purpose intrusion detection". In this context, the term "general-purpose" relates to the property of covering multiple classic as well as modern attacks rather than focusing on one or very few attack types. Another inherent problem is that benchmark data usually embrace sensitive information. This raises privacy concerns and obstruct authors of research articles from making their data sets publicly accessible (e.g. [28, 36]). This scarcity paired with regulatory impacts constitutes a delicate situation with respect to reproducibility and

comparability that certainly complicates further progress for network security research. Thus, novel quality data sets must emerge in order to relax the current state of affairs covering a broad spectrum of different attacks that overcome privacy factors at the same time.

1.2 Contribution

The contribution of this thesis is at the junction of three research directions, i.e. "network security", "machine learning" (ML) and "distributed computing". This section states the most notable contributions in these fields. We outline the main points and pinpoint required preliminary work in addition.

This work is concerned with the design and development of an intrusion detection solution to monitor network systems. Its conceptual architecture basically relies on a hybrid approach combining misuse and anomaly detection techniques based on ML principles with the clear vision to get over cumbersome packet analysis. The main contribution underlying this system can be summarized as follows:

- **Reliable flow-based detection:** As opposed to packet analysis, we expect our system to consume flows as primary data source captured in the network infrastructure to monitor. This requires valuable flow attributes at hand to determine effectively whether a network activity is intrusive or benign. Therefore, we carry out a flow feature analysis upfront. It is based on a reference implementation of IPFIX to expose several meaningful and minimal flow attribute subsets that are incorporated into our system. Additionally, we investigate supplemental information to increase detection performance while reducing false alarms. To our best knowledge, it is the first systematic approach in the domain of network security to seek for decisive characteristics covering a broad attack spectrum using flows realized through a single detection system. In this context, we debate the essential question whether general-purpose detection is possible based on empirical results employing our flow-based solution.
- **Explainable decision-making:** Predictions made by a ML system are usually difficult to comprehend by humans. Thus, one cornerstone of our intrusion detection solution is to provide transparency in the sense that care is taken to assist operational staff with follow-up activities on raised alerts by providing actionable indicators. These indicators express regularities in the data in the form of "patterns". Additionally, we permit to generate patterns for benign data. Such a transparency is not axiomatic for hybridizations and it is certainly not for conventional anomaly detectors as pointed out earlier. To incorporate the functionality behind the transparency aspect, a new pattern building algorithm is devised. It is not only unique due to its capability to learn certain and uncertain patterns in an incremental manner. Its implementation is also explicitly designed to run inside relational "database" (DB) systems making use of their efficient data structures and parallel algorithms.
- **Network dynamics:** The motion of data produced by realistic network environments is rarely discussed in network security literature. However, its consideration

is of enormous practical relevance as it affects the analytical dimension of an IDS in two ways. On the one hand, increasing workloads need to remain manageable by involved analysis techniques. Therefore, we employ established frameworks and DB systems relying on parallel and distributed computing to enable an efficient processing in terms of pattern matching, anomaly detection, pattern building and their deployment. Within this constellation, scalability is realized in which growing data volumes and velocity can be compensated by appending more hardware resources. On the other hand, the data generation process of a network may change over time such that learned ML models become obsolete. Thus, "adaptivity" mechanisms for both the pattern builder and the anomaly detector are elaborated. Connecting their capabilities goes beyond related general-purpose solutions because our system can handle different change scenarios with moderate supervision. This is enabled by an implicit "resolution process" that anticipates uncertainty in addition.

With regard to the combination of these aspects and the associated functional scope, a prototypical implementation is assembled resulting in a "hybrid flow-based IDS" (HFIDS), which exceeds the capabilities of related hybridizations. From this point of view, it can thus be considered novel. Yet, further efforts are noteworthy without which the main contribution would not be viable in concrete terms, i.e. conducting empirical assessments and handling uncertainty. These preliminaries can be highlighted along the following two points, which can be considered as isolated contributions to the respective research community:

- **Benchmark data:** There is an urgent need for quality data sets in network security research. Therefore, we put forward new network traces with a focus on the generation of malicious footages. These can be fused with legitimate traffic of one's own network site and are made publicly available without raising privacy concerns. Generally, this composition carries a broader attack range than other related data sets and its fine-grained format supports a bunch of options to evaluate competing detectors or to elaborate new concepts. Herein, these benchmark data open up the opportunity to empirically assess essential aspects of the HFIDS that would not be possible otherwise.
- **In-DB rough set model:** As formal framework for the management of uncertainty, "rough set theory" (RST) is implemented inside relational DB systems. This realization has several benefits over existing rough set software solutions including a low computational complexity and minor communication overhead. These and native qualities are utilized at two points during the construction of the HFIDS. Concretely, we employ this theory for the identification process of important flow characteristics and it constitutes the anchor for the pattern building algorithm.

In parts, the stated contribution was already published elsewhere within the scope of workshops, conference proceedings or it appeared as book chapter in order to document respective research progress. Consequently, content of this work that is based on earlier publications is largely rewritten, partly extended and annotated at the beginning of each corresponding chapter.

1.3 Outline of the Thesis

This thesis is organized along four parts. In what follows, we give an outline of each individual part with enclosed chapters.

Part I: As this work is founded on different scientific fields, we outline several important fundamentals of these disciplines in this part. Specifically, we deal with the practical capturing of flow data from networks and describe a taxonomy of IDSs. Moreover, we introduce the basics of ML and outline aspects of distributed computing (Chapter 2).

Part II: This part is devoted to the quantification of uncertainty. Therefore, we present RST as general framework to address this duty (Chapter 3). Moreover, this theory is ported to the domain of relational DBs to obtain a lightweight implementation with the capability to scale (Chapter 4). It is utilized in the main part eventually.

Part III: As main part of this work, the journey to build up the concrete HFIDS begins herein by first traveling to critical aspects that influence and shape the overall design of the system (Chapter 5). To serve the building process, one factor is the absence of available test data, which provides sufficient motivation to build a new benchmark data set (Chapter 6). It is utilized next to perform an in-depth analysis to expose meaningful flow features (Chapter 7). In the remainder of this part, we concentrate on developing an anomaly-based method (Chapter 8) and on implementing the pattern building algorithm (Chapter 9). All partial results up to this point are finally assembled to deploy the HFIDS followed by an intensive evaluation completing the journey (Chapter 10).

Part IV: The last part closes this thesis. It first concludes on the achievements elaborated in this thesis and discusses opportunities that can be leveraged to improve the resulting HFIDS. This includes the notion to elevate parts of the system towards a collaborative solution (Chapter 11).

Chapter 2

Background

A general overview to important aspects of network security, ML and distributed computing is provided in this chapter. We start by introducing a modern technique permitting to capture network traffic efficiently based on flows and present a skeleton to categorize systems design to uncover intrusions in practice. Furthermore, the basic conception behind ML is illuminated that is relevant for this work. In this light, we also debate inherent problems jeopardizing the applicability of ML for many real-world scenarios. Lastly, a wrap-up of distributed computing is worked out in the context of "big data". We outline the basic challenge masked underneath big data, give insights to two reference architectures for big data infrastructures, concretize a stream processing framework and discuss the fundamental concepts to build up parallel DB systems.

2.1 Flow Capturing and Intrusion Detection

The monitoring of traffic to analyze and safeguard a network system is both essential and critical at the same time. For this reason, we briefly introduce a modern approach to capture network activities in a standardized fashion offering an abstracted view to network traffic (Section 2.1.1). Independent from this initial objective, a taxonomy is outlined to categorize intrusion detection solutions along three distinguishing dimensions in order to provide an overview of opportunities to uncover malicious and illegal activities (Section 2.1.2).

2.1.1 The Flow Standard IPFIX

"Network management" is the process concerned with the administration and controlling of network systems. This includes fault analysis, accounting, performance management and other important aspects to guarantee "quality of service" (QoS). Enablers to pursue these management tasks are techniques to collect information from the operating networks. Traditionally, they rest upon full packet capturing to monitor traffic activities or management protocols such as the Simple Network Management Protocol (SNMP) to control and maintain the operability of network components directly. Another approach that grew out of these attempts is to observe network systems based on the notion of flows [37]. Standardized by the Internet Engineering Task Force (IETF) in 2013, a flow can be described as "*... a set of [IP] packets or frames passing an observation point in the network during a certain time interval [and] all packets belonging to a particular flow have a set of common properties*" [7]. By this means, flows abstract one up to thousands of packets, which provides several practical advantages. One obvious benefit is weaker storage requirements compared to packet data but we are detailing the assets of flows in a later chapter. Instead, our focus herein is to give a brief summary of the IETF flow standard IPFIX [7] that evolved from NetFlow version 9 [8], a protocol supplied by Cisco Systems Inc.[1]. Additionally, we outline some subsequent "request for comments" (RFCs) that are related to IPFIX.

The process to render flows from available network traffic essentially can be brought down to three main stages: (i) packet observation, (ii) flow metering and exporting and (iii) data collection and analysis as given through Figure 2.1. In stage (i) packets are tapped at one or multiple defined observation points strategically situated inside the network system of interest. The two common techniques to implement traffic capturing functionality are to leverage existing forwarding devices in the network using "port mirroring" or to place a sensor between two network components gathering the traffic "in-line". From there, extracted packets are sent to the next stage (ii) that comprises the metering and exporting. The metering takes care of the flow creation and finally determines when a flow is forwarded to the exporting. Thus, stage (ii) can be considered as the cornerstone of the IPFIX standard. To keep track of the flow creation, the metering relies on the conception of a "flow cache". Yet, before we can elaborate on the functioning of that central data structure, let us get back to the common properties

[1] `https://cisco.com`

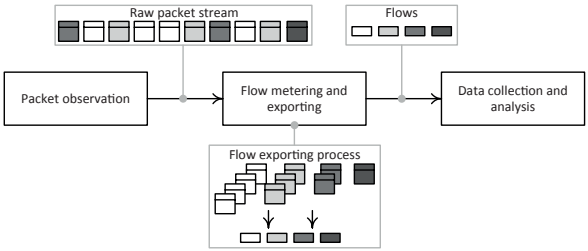

FIGURE 2.1: Main stages of common flow exporting

postulated as part of the flow definition above. What is meant by these properties is a composed "identifier" (ID) that is termed "flow key" in IPFIX jargon. It typically consists of the five-tuple

$$\langle \text{src. IP address}, \text{dst. IP address}, \text{src. port}, \text{dst. port}, \text{transport protocol} \rangle \ ,$$

but also other information directly or indirectly derived from packets is conceivable to shape that key. As such, the mechanism behind the flow cache can be understood as plain "hash table" where a bucket reflects a flow entry and the related flow key is the hash key. With this structure in place, it can be determined efficiently for each arriving packet whether an active flow already exists in the cache to which the packet can be associated or a new flow entry is appended otherwise. We also have to consider the information collected during the metering process. This is encapsulated by "information elements" (IEs) according to RFC 7012 [38]. They specify any field that can be exported from a flow including key fields. Non-key IEs are, for instance, the number of packets or octets carried by a flow. All common ones are organized along their IE ID, data type, description and other attributes in the Internet Assigned Numbers Authority (IANA) registry for IPFIX entries[2]. Apart from this global repository, IEs can be modified or new ones can be defined dependent on individual demands. Such customizations are encoded by means of IANA "private enterprise numbers"[3] (PENs). Furthermore, the metering is also in charge of determining when a flow terminates. RFC 5470 [39] describes three expiration scenarios: if no packet has been seen for a flow in a specific time period, a flow resides for a longer time in the cache due to the continuation of corresponding packets or resource exhaustion occurs, a "flow expiration" is identified. The former two conditions are dependent on thresholds to which we refer as "idle timeout" and "active timeout". According to this, they need to be configured for the metering process upfront. Beyond these expiration policies, "natural expiration" is also established in practice as discussed in [6]. Similar to a conventional ending in the sense of a Transmission Control Protocol (TCP) connection, a flow termination is signaled if the TCP flags FIN or RST have been observed. Once any of the mentioned expirations is detected, the corresponding flow entry is removed form the flow cache and handed over to the exporting process. The main purpose of this process is to build "IPFIX messages" that encapsulate completed flows and their structure according to the metered IEs sending them to the data collection and

[2] `https://iana.org/assignments/ipfix/`
[3] `https://iana.org/assignments/enterprise-numbers/`

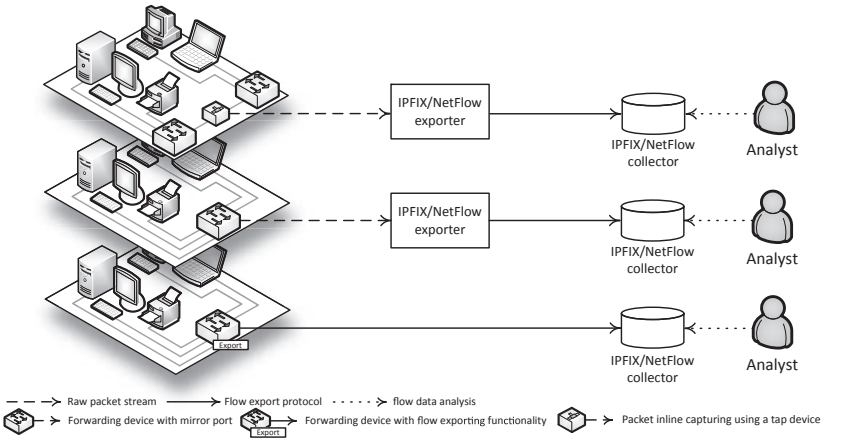

FIGURE 2.2: Different configurations to capture flows from networks: packet in-line capture with dedicated export and collection (top), packet capture using a mirror port with dedicated export and collection (center) and packet capture and flow export combined in a forwarding device (bottom)

analysis via a selected transport protocol. This last stage (iii) basically consist of one or multiple "IPFIX collectors". These collectors should have the capability to accept and decode IPFIX messages that are either buffered in-memory for the consumption at some other application or persisted permanently in a data storage permitting analytics. After pointing out these main stages, it should be noted that the metering and exporting are treated as isolated processes in the IPFIX standard. Yet, they are tightly connected in practice and so we refer to any software or hardware system implementing both processes as "flow exporter". A picture of common configurations to render flows from a network system is illustrated in Figure 2.2

When reviewing the typical five-tuple of the IPFIX standard, it immediately follows that flows are unidirectional per definition, i.e. at least two flows are generated for a conventional TCP connection. However, there are several applications where it is more appropriate to look at the data flowing between two endpoints in one go. This is anticipated by RFC 5103 [40] introducing "bidirectional flow" (biflow) semantics. In simple terms, a biflow can be understood as the fusion of two associated unidirectional flows. Consequently, biflows contain more IEs compared to their unidirectional counterpart because directional IEs such as the number of packets or octets have to be recorded for either direction. Other specifics of the IPFIX protocol also exist but were neglected in this section for the sake of brevity. These include optional functionalities such as sampling and filtering that are intermediate steps between metering and exporting or the mediation between exporting and collection. Therefore, we refer to [37, 6] or to the respective protocol documentations [7, 39, 41] directly in order to get a good entry point for further reading. Other practical aspects along the benefits of a standardized process to monitor network systems and a discussion of several existing software-based flow exporters can be found in Chapter 5.

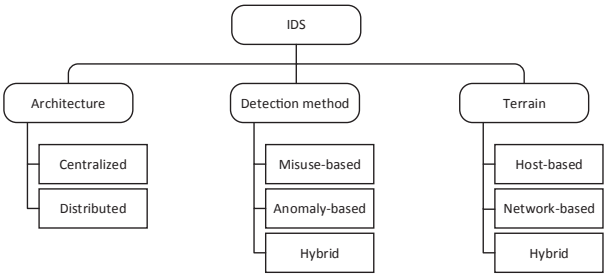

FIGURE 2.3: Basic IDS taxonomy along three dimensions

2.1.2 A Taxonomy of Intrusion Detection Systems

According to [42] reflecting one of the seminal works in the context of cybersecurity, a threat can be defined by unauthorized attempts to access or manipulate information or by restraining the operability of a system. This early version to specify unsolicited actions not only provides an intuitive description but also documents the desire to detect such intrusive behaviors from the beginning of the 1980s onwards. Consequently, a variety of IDSs have evolved with different focus areas since then. Among the peculiarities each system carries, there are at least three purposive dimensions to narrow down their main characteristics. They are outlined in Figure 2.3 and briefly discussed in what follows.

Let us start with the perimeter the IDS should observe. Basically, there are two choices. Either the system is intended to analyze an individual computer system by monitoring its runtime with current processor utilization, memory consumption, ongoing system calls and audit files or it is in charge of inspecting an entire network system based on available traffic information. We refer to both types discriminating the operational terrain as host-based IDS (HIDS) and network-based IDS (NIDS). HIDSs are advantageous when there is ambition to gain detailed insights into a single host. The vision of a NIDS, on the other side, is broader as multiple computer systems can be observed concurrently. Yet, NIDSs are limited in the sense that they only can detect illicit actions if they are reflected in the network traffic and so malicious behavior that remains local cannot be dismantled by definition. To get over these downsides, hybridization attempts are conceivable. For instance, one can think of multiple HIDSs that are interconnected to shared audit information or an IDS capable to fuse and process data sources on multiple levels. Two pioneering HIDSs are IDES [43, 44] and MIDAS [45] whereas IDES is also able to examine audit trails on a global level in order to circumvent isolated treatments. NSM [46] in turn is the first IDS employing network data as primary data source. An outgrowth of this system is DIDS [47] that also incorporated host agents to record notable events at a local level to reduce blindspots. A further categorization of IDSs follows implicitly from the mentioned hybridization relieving terrain restrictions, i.e. the distinction on an architectural level. In this regards, IDSs can be either centralized or distributed. In their most basic form, HIDSs and NIDSs are centralized but can evolve towards a distributed system if they engage multiple interacting components. Good examples proceeding this way are IDES and DIDS according to their enhanced

functionality whereas the processing of NSM is basically central. There are a bunch of other distributed IDSs that can be found in literature. A compilation of the most influential systems towards "collaborative intrusion detection" is outlined in [48]. The last dimension we would like to discuss was already touched in Section 1.1 where we distinguished by misuse-based and anomaly-based intrusion detection approaches. We recall that misuse methods deal with the specification of a model that expresses malicious behavior and predominant realizations of this type are "signature-based systems". In this respect, a signature can be understood as a form of rule describing the underlying "attack pattern" by a series of conditions that, once fulfilled, fires an security alarm to unveil potential system interferences. Well-known representatives of this sort are P-BEST [49] and Snort [50]. Orthogonal to misuse detection are anomaly-based approaches. They are basically concerned with the definition of a profile that is capable to describe legitimate and normal activities. The conception behind this technique was originally proposed in [43] as part of IDES and many other anomaly detection developments followed this perception. Two of them are SPADE [51] or PAYL [52]. Yet, besides the statistical anomaly component comprised within IDES, there is a remarkable design choice to it as P-BEST is employed in addition, which showcases the first hybrid approach in terms of detection. Instead of favoring one detection approach over the other, their fusion has several amenities, which are experienced during the course of this work. To get an overview of related work in this direction, we refer the interested reader to Chapter 5.

2.2 Machine Learning

The branch of computer science that studies computer programs capable to raise their performance on a particular task as more experience is gathered is termed ML [53]. Given this rather broad draft, one can anticipate countless use cases for ML in nearly every sphere of our daily and future life starting from smart watches that predict the necessary next steps to improve one's health situation in a sustainable fashion up to self-driving transport services dropping passengers at their desired destination right on time according to the most optimal route. Beyond that, network security is another domain where ML plays a central role as it provides advanced support to expose unsolicited cyberactivities. Therefore, several general aspects of ML are outlined in this section. We start by distinguishing between the two most common learning setups of ML, sketch basic considerations and also briefly highlight situations that are at their intersection (Section 2.2.1). On these grounds, a lack of sufficient expressiveness is discussed (Section 2.2.2). Furthermore, an inherent problem is detailed refusing conventional ML methods from being effective in many real-world scenarios (Section 2.2.3).

2.2.1 Supervised and Unsupervised Learning

The wellsprings of ML date back to the middle of the last century (e.g. [54, 55, 56]). Since then, ML research has evolved in many directions and, thus, holds a variety of peculiarities dependent on the context, which makes a sharp distinction of its applications rather difficult. However, an attempt can be made to roughly organize its key concep-

tion around two learning scenarios that are essential for most parts of this work, i.e. (i) "supervised learning" and (ii) "unsupervised learning". Consequently, we summarize important matters of both types in this section and also state learning strategies that are at their junction.

From an abstract point of view, a supervised learning task can be seen as an approximation of an unknown target function $c : \mathcal{V} \rightarrow \mathcal{C}$ that maps examples x from an instance or input space $\mathcal{V}$ to concepts or classes in $\mathcal{C}$. To solve this task, we are given a finite set of n examples for which we know the true target value $c(x)$ with certainty given by a teacher or domain expert. This collection of data is commonly referred to as "training set" and can be denoted by $T = \{\langle x_1, c(x_1)\rangle, ..., \langle x_n, c(x_n)\rangle\}, n \in \mathbb{N}$. Then, the ultimate goal is to find a "hypothesis" h based on T such that $h(x) = c(x)$ for virtually every example that comes along. This also includes unseen observations for which the target value is generally unknown and, thus, has to be predicted henceforth. We refer to the period where a learned hypothesis is applied to unseen data as the "live phase". As such, supervised learning is an "inductive learning process" (e.g. [53, 57]) where we try to find a generalized description from examples or facts explaining them. We also name the hypothesis h the "model", "theory", "classifier" or "predictor", which is induced by an algorithm, i.e. the "learner" or "inducer". Common supervised learning algorithms are decision trees (DTs) (e.g. [58, 59]), artificial neural networks (ANNs) (e.g. [54, 60]), rule-based approaches (e.g. [56, 61]) or support vector machines (SVMs) (e.g. [62, 63]) stating only a few. The learning task of interest can be determined by the codomain $\mathcal{C}$ of the target function c. It is usually known a priori as it dictates the number of applicable algorithms. In case $\mathcal{C}$ is continuous, the task refers to a "regression problem", while a finite number of discrete values points to a "classification problem". In this work, we mainly concentrate on binary (i.e. $|\mathcal{C}| = 2$) and multiclass classification problems (i.e. $|\mathcal{C}| > 2$) unless explicitly stated otherwise. Hence, we call a known target value $c(x) \in \mathcal{C}$ the "class label" or "decision" for example x and refer to this data as "ground truth" (GT) when considered as collective information. Within this framework and with the absence of any further information, the fundamental assumption of supervised learning is that a found theory performing well on sufficient training data is also believed to have a solid performance on unseen examples, i.e. the "inductive learning hypothesis" [53]. To estimate these generalization capabilities in practice, a learned model, therefore, is commonly examined against a test set of instances with underlying GT that either coexists or that is extracted from available training data right before learning commences. If the model is very complex fitting training data just right but performing arbitrarily on the test set, it is said to "overfit" the training data and has high variance. Despite its complexity, it obviously generalizes not well enough beyond familiar observations, which leads to an established paradigm known as "Occam's razor" named after William of Ockham[4]. In essence, it states that one should prefer the simplest hypothesis explaining the data over any other produced theory to reduce overfitting. Yet, it should not be too simple because this would results in another extreme called "underfitting". This phenomenon in turn is caused by theories that are too rudimentary to describe the underlying structure of the data inferring too strict assumptions that were taken to approximate the target, i.e. a high "inductive bias". Thus, we are eager to find a balance

[4] William of Ockham was an English Franciscan friar and philosopher who debated about the principle of parsimony around the year 1320.

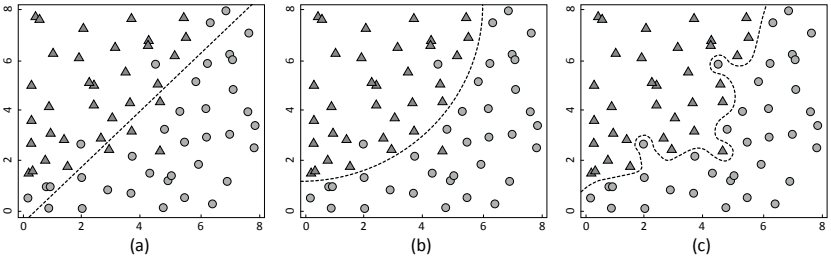

FIGURE 2.4: Decision boundaries of three different models to solve a binary classification problem on a two-dimensional instance space: (a) underfitting, (b) adequate fitting and (c) overfitting

between both error sources, which is referred to as "bias-variance trade-off" in literature (e.g. [64, 65, 66]). A qualitative representation of this trade-off is illustrated in Figure 2.4 where the task is to solve a binary classification problem. The model in Figure 2.4 (a) is too simple to describe the data well, while the theory pictured in Figure 2.4 (c) is far too complicated. An intuitive generalization is provided by Figure 2.4 (b), which accepts a minor proportion of misclassification in order to capture the natural structure of the data.

As opposed to supervised learning, there is no teacher in an unsupervised learning setting and the class assignment of examples is not known consequently. On this account, the task comprised is not to learn a classification in the first place, but to find group-like structures. With this limited set of information, unsupervised learning turns out to be rather difficult as there is no indicator at our disposal to evaluate whether sought and exposed structures are acceptable or not (e.g. [66, 67]). However, it has strong practical relevance. Suppose we are in charge of developing an existing business further and are given the volume of customer data containing transactions of purchased items along with personal data. Then, it is of strategic importance to answer questions such as: Which type of customer segments exist in the business? Which segment refers to the most loyal buyers? What is the target group of a specific product? To respond to this kind of questions, one does not require a teacher necessarily but a method that is capable to separate customer data from another according to their intrinsic characteristics. One approach to achieve this goal is commonly referred to as "cluster analysis", which is the most common task applying unsupervised techniques [68]. It finds distinguished sets or hierarchies in the data according to some distance-related quantity and is often divided into two broad categories, i.e. "partitional algorithms" and "hierarchical algorithms" (e.g [69, 70, 71]). The main characteristic of the former type is the ability to separate the instance space into disjoint groups, which means that each particular example belongs to exactly one cluster. A prominent example for this type is the k-Means algorithm. In order to get an adequate grouping, it initially places $k \in \mathbb{N}$ "centroids" in the data space at random. Based on that, it tries to minimize the "sum of squared errors" between data points and the centroids and leverages this grouping to recompute the centroids iteratively until the process converges, i.e. there is no notable change to the centroids. As a result, the data space is partitioned into k

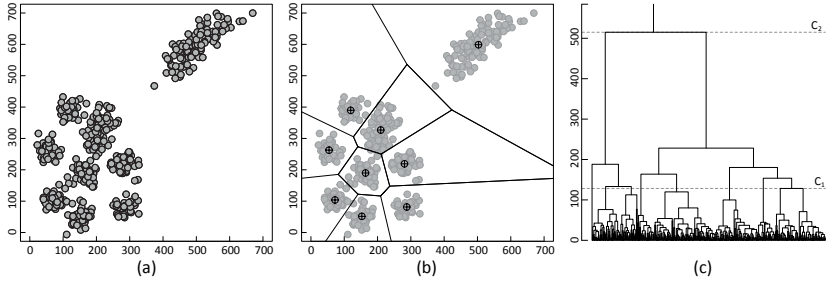

FIGURE 2.5: Seeking the structure of a two-dimensional instance space: (a) original unlabeled data (●), (b) partitional clustering using k-Means ($k = 9$) with centroids ($\oplus$) represented as Voronoi diagram and (c) the result of a hierarchical clustering depicted as dendrogram; cut-off point C_1 of the hierarchy showcases the same outcome as k-Means whereas C_2 is another valid segmentation when reviewing the data from a higher level, i.e. the emergence of only two clusters

non-overlapping clusters. Beside this approach, several other alternatives emerged such as given in [71, 72]. Depicting the EM algorithm, processing works quite similar to the classic k-Means but clusters are described in terms of mixture models (MMs), i.e. the attempt to model data using probability distributions. This statistical representation permits to detail clusters in more depth as opposed to the pure centroid representation of k-Means and enables group overlappings, which gets over a hard cluster assignment that can be an issue in addition. Another concern to this type of algorithms are applications where the number of clusters are unknown in advance and so there is a problem to set the input parameter k accordingly. Yet, a remarkable number of attempts can be found in literature to estimate the amount of groups in practice (e.g. [73, 74, 75]). Hierarchical algorithms in turn completely ignore such a parameter because their primary goal is to build a cluster hierarchy, i.e. a composition of subclusters. Hence, it can be understood by a conventional tree structure where leaf nodes represent individual observations, higher nodes constitute the parent cluster of lower nodes (see subclusters) and the root node is the universal cluster. Approaches seeking for such an assembly can be broadly categorized by their search mechanism, i.e. "agglomerative clustering" and "divisive clustering" (e.g. [69, 71]). While the former method tries to build up the structure bottom-up, the latter conducts a top-down approach. Even though there are several applications for hierarchical clustering, they are rather inappropriate for larger data sets due to their computational complexity and space consumption. Consequentially, further elaboration in this direction is skipped. Instead, we relegate the interested reader to relevant literature such as [69, 70, 71]. Yet, an illustration of both partitional and hierarchical clustering approaches is depicted in Figure 2.5 using the well-known da1 data set [76].

Additionally, we want to stress that supervised and unsupervised learning are not necessarily disjoint and tasks at their intersection simply can arise if one is confronted with both labeled and unlabeled data. Learning in this terrain is very appealing because the labeling of data in order to train a supervised learner is a cumbersome and time consuming exercise for many sophisticated learning setups including our problem domain

(e.g. [13, 77, 78]), while the acquisition of unlabeled data is usually easier to accomplish. Thus, one is eager to build a predictive model that has better classification capabilities than a theory made up of pure labeled data by exploiting the structure of unlabeled data in addition, which yields an active area of research, i.e. "semi-supervised learning" (e.g. [79, 80]). Early strategies of semi-supervised learning include "self-training" [81] and "co-training" [82]. Employing self-training, a theory is created based on available labeled training data that is utilized to predict the classes of unlabeled data. So far, this setting corresponds to a standard supervised learning task but it does not terminate at this stage. Moreover, it leverages those made predictions to expand the amount of labeled data for subsequent training rounds. Co-training, at the other end, trains a supervised learner on varying attribute subsets[5], which renders different models due to their different perspectives on the data. Hence, most confident predictions can be shared among the models to essentially enhance the labeled training set and teach each other ultimately. It assumes that those subsets are sufficiently expressive and conditionally independent. Other approaches leverage both data sources simply by applying an unsupervised learning algorithm to identify groups in the data that are assigned to the most probable class determined by labeled instances directly or by a learner trained on them (e.g. [83, 84]). An alternative to such strategies, often termed "cluster-then-label" approaches, are MMs. For instance, [85] uses a variant of the EM algorithm to identify structures in the data for text classification tasks. The authors demonstrate empirically that predictors built based on labeled and unlabeled data can have a better classification performance than theories created on those pure labeled data. Similar results are shown in [84]. Yet, it is not guaranteed that the outcome of semi-supervised learning is always better than supervised learning. In fact, it relies on assumptions as any other ML approach to work effectively. Here, we depict two famous ones, i.e. the "smoothness assumption" and the "cluster assumption". The former basically states that two close data points in a high-density region are very likely to share the same class label, while the latter claims that data points in the same cluster belong to the same concept. These and further advanced prerequisites are discussed in [79]. Lastly, we would like to briefly mention a further learning paradigm that is closely related to semi-supervised learning as it also tries to make the most out of both labeled and unlabeled data. It is called "active learning" (e.g. [86, 87]). In addition to semi-supervised learning, a special capability is enclosed though. The learner in charge has the freedom to issue queries for unlabeled examples to unveil their true class labels that are answered by an "oracle" (e.g. a human expert). With this great advantage at hand, the objective of active learning is to build a predictive model that outperforms a conventional learner by using significantly fewer labeled data than required in a typical supervised learning setup. Therefore, different querying strategies can be applied to maximize classification performance and to bother the oracle as little as possible. Two common ones are "uncertainty sampling" [88] and "query-by-committee" [89]. While in the former strategy, the oracle is asked for labels of instances the trained learner is least confident about, the latter considers multiple learners and queries for labels of those unlabeled examples that are situated in controversial regions (i.e. data in regions resulting theories disagree). Using these or other strategies, many empirical evaluations show that the objectives of active learning can be achieved. This is supported by the survey in [90]

[5] We refer to Chapter 3 for a rather formal consideration of attributes and selecting subsets of them to solve a classification task.

where researchers were asked about the suitability of active learning. It was found out that expectations about active learning were met partially or fully confirmed by nine out of ten participants. However, it should be stated that this survey mainly dealt within the discipline of "natural language processing" and so its outcome indeed biases in this direction. Further positive outcomes in other problem domains can be found in [91, 92], for instance.

2.2.2 Model Transparency

In the previous Section 2.2.1, we introduced supervised learning as an inductive learning task that seeks a generalized description explaining well both existing and unseen data. One of the problems with this objective is that even though predictive performances are oftentimes solid, many learning algorithms produce opaque and obscure models that are very difficult to understand by humans. Considering ANNs for instance, they try to emulate the structure of the human brain where interconnected artificial neurons are the centerpieces of each network. Organized by one or multiple layers, input data are consumed by the network and a prediction is created according to the firing of specific neurons that in turn is basically realized by computing "linear combinations" of the input and weights. Employing this conception endows to model arbitrary complex relationships between input and output variables but as the number of neurons and/or the number of layers is increased, reproducing the cause of a taken decision becomes harder too. Dependent on the network structure, one often would have to trace back millions of calculations that led to the actual prediction in the network. Obviously, this is an infeasible undertaking for a human being as part of a post-hoc analysis. SVMs are another example where sophisticated classification tasks can be modeled. In the most basic form, they attempt to build a hyperplane that maximizes the margin between the closest opposing data points in the training set, i.e. the support vectors. From there, the hyperplane is utilized to distinguish between one or the other class. While follow-up activities on the model are already difficult to comprehend when the instance space has two or three dimensions, higher dimensions ultimately exceed our spatial skills rendering SVMs inappropriate to provide an understandable decision. A similar argumentation holds for probabilistic models such as MMs. We name such non-transparent and complex models "black boxes", which is a common metaphoric term in literature (e.g. [93, 94, 95, 96]) to differentiate them from other more intuitive attempts such as instance-based models, DTs or rule-based classifiers. In this respect, an essential question is what are the differences of these more intelligible ML model families. Perhaps their secret ingredient stems from their transparent design where the costs to figure out a taken decision are intrinsically reduced and no deep technical understanding is required necessarily. In the case of instance-based learning, we are eager to predict the class of an unseen example by simply looking at familiar data points given during training instead of building an incomprehensible model. As such, we have concrete facts at hand as a reference that can guide us to get a clue about the behavior of a classifier. Suppose an object in question is associated to a specific class, then we can explain that prediction because out of the most similar objects found in the training data, say five, four also share this class assignment. Additionally, we can inspect these five representatives in more depth and distill those peculiarities that potentially led to the classification result

and separate those that are rather irrelevant for the decision. Yet, this final observation to unveil similarities and dissimilarities is not directly supported by basic instance-based models. DTs, on the other end, provide such an analysis implicitly. As their name infers, a model is constituted by a tree-like structure where nodes represent characteristics, descending branches are their concrete values and leaf nodes correspond to decisions. This way, we get an comprehensive description of the produced theory, which can be explored intuitively where nodes situated at a higher level in the hierarchy are usually more influential for the decision-making than those close to leaf nodes. To reconstruct a made prediction, we simply have to traverse the tree according to the value of an object from its root down to the leaf and also can inspect alternative paths quite naturally, which requires manual intervention for raw instance-based models. Critical to DTs, though, is their size that often is unintentionally large. It is an indicator for overfitting and certainly constrains interpretability (e.g. [97, 98]). Rule-based models can get over this dilemma. They consist of a set of "decision rules" each having the form:

IF *condition* THEN *consequent* .

The condition is a logical expression that, once true, leads to the consequent. Despite the obvious intuitiveness of decision rules, their conditions can take arbitrary forms. This is beneficial because, unlike DTs, the underlying logic is not squeezed in a unified data structure. Considering a produced rule set where none of the rules are overlapping in terms of their conditions, it is cumbersome to build a DT with the same equivalent meaning as no node can be shared effectively. To rectify such situations, we would have to implant redundant structures to the tree, which impacts the tree size and interpretability aspects needlessly. This problem is commonly knows as the "replicated subtree problem" making rule learning both more transparent and compact at the same time in comparison to DTs (e.g. [61, 93, 99]). Furthermore, literature frequently argues that rules have an advantage over DTs due to their improved human readability (e.g. [53, 93, 100]). Consolidating these findings with the view of other authors (e.g. [49, 101]) leads to three key aspects favoring rule learning when transparency is desired: (i) rule-based models organize extracted knowledge by a conventional set and each element in that set represents a piece of atomic knowledge. (ii) The logic comprised by rules is encapsulated by natural language constructs and, therefore, easy to understand expressing the reason for a taken decision implicitly. (iii) Each rule is self-contained and can be viewed in isolation. As such two important properties can be derived. On the one hand, it is not required to understand the entire model in exchange for getting the explanation of a single prediction in question. Hence, it abstracts away complexity. On the other hand, this modularity is expedient as knowledge can be extended or replaced straightforwardly. To illustrate the taken discussion on comprehensibility, Figure 2.6 attempts to categorize the mentioned model families according to their intrinsic transparency.

Lastly, we want to emphasize that the lack of interpretability in ML is a serious problem with enormous practical relevance. As more critical applications make use of ML, there is a growing need to gain insights into the machine's decision strategy. This trend, for example, was anticipated by the European Union's "right to explanation" as part of the

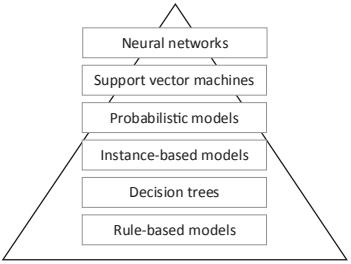

FIGURE 2.6: Model transparency pyramid: comprehensibility increases as one slides down the pyramid from top to bottom; contained list of model families are not exhaustive (adapted from [93])

European General Data Protection Regulation[6] that took effect in 2018. In essence, it postulates the right of citizens to receive a meaningful explanation once involve in an algorithmic decision. Further evidence for the demand of transparency in predictive modeling comes from the scientific community that, in fact, established a whole new line of research in most recent years basically distinguishing between "interpretable ML" and "explainable ML". While the former movement attempts to investigate algorithms that provide off-the-shelf interpretability, the latter primarily concentrates on "model-specific" or "model-agnostic" techniques. Model-specific method typically try to leverage particularities of a specific black box predictor to derive interpretations and model-agnostic approaches generally train a transparent model to explain any black box at a local scope (e.g [94, 96]). Based on the discussion in the previous paragraph, it is not surprising that model-agnostic methods oftentimes fall back on rule or DT learning principles. The prominent LIME framework [95], for instance, is a good representative for model-agnostic approaches.

2.2.3 Nonstationary Environments

Even though we are given a huge amount of historic data that can be exploited to estimate trends as part of a forecasting tasks, we are unaware of the future in the general case. This point is also inherently affecting ML and addressed by the inductive learning hypothesis reviewing Section 2.2.1. It is central because it anticipates the structure of unseen instances from available training examples. This essentially means that both training set and data in the live phase are expected to be drawn from the same, albeit unknown, distribution. While this assumption certainly holds true for several disciplines, it is rather idealistic for many real-world scenarios and promptly becomes prohibitive when data are collected over a long time frame and involved variables of the underlying problem domain are subject to change. Examples of such situations are very common and vary widely. These include the aging effects of industrial sensor devices that are involved in the data generating process [102], dynamics of user profiles in computer systems [103] or marketing applications where customer habits might be influenced

[6] `https://privacy-regulation.eu`

by advertisements or fashion trends [104]. Another example are long-term studies of medical data that are collected over years or even decades [105]. Thus, it is very likely that the data distribution may have changed over time making "data mining" tasks a difficult endeavor. This shift from stationary to nonstationary environments in ML does not necessarily indicate a problem as long as the learned theory remains valid, i.e. it is still able to model live data adequately. Yet, it becomes a serious affair if the fluctuation affects the relationship between instance space and one or more target concepts. As a consequence, the classification quality of a trustworthy model can degrade drastically, which may yield a useless theory eventually. This phenomenon is referred to as "concept drift" in recent literature (e.g. [102, 106, 107, 108]) and evolved from the seminal works in [109, 110] by the mid 1980s. Since then, solving problems in that direction received relative few attention for the following two decades but with the advent of "stream mining"[7] in the mid 2000s (e.g. [68, 111, 112]), learning in nonstationary environments ultimately has become a hot topic up until now. This is due to the nature of data availability in streaming problems as they evolve over time. Thus, changes can occur unexpectedly and are unpredictable, which is the general assumption operating under drifting circumstances (see [106, 107]). In this respect, one can distinguish between two types of drifts, i.e. "virtual concept drift" and "real concept drift". Both types embody different characteristics, which can be summarized by leveraging "Bayesian decision theory" (e.g. [113, 114]) in terms of "what" is actually changing. Using this theory, we are able to express the classification of an example x to class label $c_j, 1 \leq j \leq k \in \mathbb{N}$ employing "Bayes' theorem" provided by

$$P(c_j|x) = \frac{P(x|c_j) \cdot P(c_j)}{P(x)} \ , \tag{2.1}$$

where $P(c_j|x)$ is the conditional probability of c_j given x (i.e. the posterior probability), $P(x|c_j)$ is the likelihood of x with respect to c_j, $P(c_j)$ is the class prior probability of c_j and $P(x)$ is the evidence. This way, both drift types can be explained as follows (e.g. [102, 107]): a virtual concept drift occurs if for some example x and class label c_j:

$$P_t(x) \neq P_{t+1}(x) \text{ and } P_t(c_j|x) = P_{t+1}(c_j|x) \tag{2.2}$$

between two varying time points t and $t+1$ denoted as subscripts, i.e. the probability distribution of the instance space is subject to change but the class membership is not affected. A real concept drift in turn is noticeable in those cases where

$$P_t(c_j|x) \neq P_{t+1}(c_j|x) \text{ and either } P_t(x) = P_{t+1}(x) \text{ or } P_t(x) \neq P_{t+1}(x) \ , \tag{2.3}$$

which means that the relation between the instance space and target indeed is shifting disregarding whether $P(x)$ changes over time. From this perspective, nonstationary environments should not be confused with "data noise" as it only represents insignificant fluctuation in the data that is not directly related to the actual drift. A visual depiction of these two drift types along with noise in the data is presented in Figure 2.7.

Despite of this formal introduction of concept drift in terms of changing data popula-

[7] Stream mining is often associated with the term "big data", which we illuminate in Section 2.3 with a focus on conceptual and technological aspects.

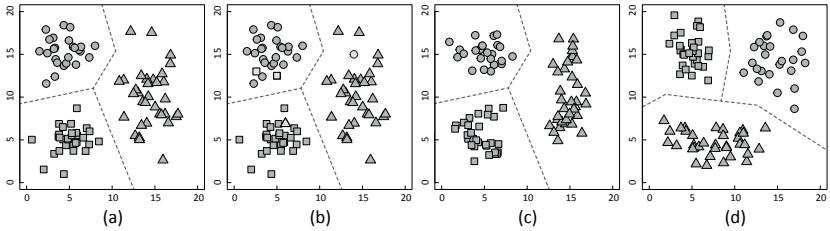

FIGURE 2.7: Nonstationary learning scenarios and data noise on a two-dimensional instance space where shapes of the examples represents their class assignment underpinned by the dashed decision boundaries: (a) original data, (b) data noise, (c) virtual drift and (d) real drift; in a virtual drift scenario, the decision boundary remains intact, while a clear change is notable within a real drift

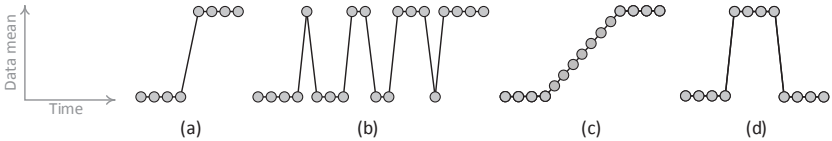

FIGURE 2.8: Schematic illustration of different concept drift facets: (a) sudden drift, (b) gradual drift, (c) incremental drift and (d) reoccurring drift (adapted from [107])

tions, we have not discussed the different shapes a drift can take, which refers to its speed or its intensity. To this end, most literature separates between "sudden drifts" and "gradual drifts". On the one hand, a sudden drift involves an abrupt change of the data generating mechanism, which is most crucial in practice because an initially trained and static inducer becomes ineffective rapidly causing high misclassifications straightaway. On the other hand, a gradual drift can be defined by a slowly evolving distribution change. It is not as critical as an abrupt drift from start but may have the same implication at the very end for an existing model with respect to its predictive performance. Sometimes, authors also refer to two other drift variants, i.e. "incremental drifts" (e.g. [106, 107]) and "reoccurring drifts" (e.g. [102, 107]). While in the former case, more than two distributions are involved to finally change one or more concepts in a stepwise fashion, the latter drift is interesting for several applications that try to model seasonality or other cyclical effects. According to [115], there is, however, controversy in literature to the introduction of incremental drifts as they appear to be similar to gradual ones. This is most probably the reason why several authors are not defining incremental drifts explicitly (e.g. [102, 116, 117]). Likewise, reoccurring drifts can be modeled using two consecutive abrupt drifts from one distribution to another and vice versa. Based on these grounds, the reoccurring drift facet is treated with less priority in this work. Yet, a schematic overview of all discussed drift characteristics is illustrated in Figure 2.8.

Given this general introduction of concept drift in the field of ML, we would like to emphasize that it is rather difficult to provide a uniform definition because of varying terminologies and inconsistencies among them that can be found in literature. This can

be exemplified by the terms "concept shift" [118], "population drift" [119] or "conditional change" [120] that refer to a concept drift in this section, while virtual drifts are also sometimes denoted as "temporal drift" [121], "sample shift" [118] or "covariate shift" [122] with and without affecting the decision boundary (e.g. [116, 123]). We relate this issue to the historic evolution of the problem from various disciplines with different focus and background. This finding is supported by other authors in addition clearly stating that there is a lack of standardized terminology (e.g. [102, 115, 124]), which ultimately leaves a detailed description of the problem and involved constraints up to each researcher. Despite these mentioned difficulties to describe a nonstationary environment properly, we stick to the definitions given above throughout this work and contextualize them closer with network security and intrusion detection in the respective chapters to follow.

2.3 Big Data Processing

The ability to process growing amounts of data is a key challenge for network security and many other disciplines oftentimes contextualized with big data. In this section, we first provide a critical discussion about big data by trying to unveil and frame its main characteristic ground (Section 2.3.1). Second, we detail the two reference models commonly utilized to build up big data infrastructures (Section 2.3.2). Lastly, we present a concrete stream processing framework (Section 2.3.3) and DB architectures (Section 2.3.4) capable to run in such scenarios.

2.3.1 The Phenomenon

The buzzword "big data" is an umbrella term that covers various challenging aspects with respect to our standard way to solve problems using computer systems. It emerged based on the needs to deal with the continuous growth of data "volume", "velocity" and "variety" that goes back to debates of the early 2000s. In this context, "volume" refers to the amount of data to process while "velocity" aims at the speed of data arrival. Lastly, "variety" stands for the structure of the data that can be very loose, tight or a mixture of both dependent on the application. Years later, these demands commonly known as the "three Vs" became also the cornerstones of the famous definition of Gartner Inc.[8] stating that "*big data is high-volume, high-velocity and/or high-variety information assets that demands cost-effective, innovative forms of information processing that enable enhanced insight, decision[-]making, and process automation*" [125].

Yet, the three Vs are not the only definitions that exist. There is a bunch of further proposals ranging from the expansion of the Vs (e.g. [126, 127]) to the incorporation of cultural aspects (e.g. [128]) and so there is no homogeneous and commonly accepted definition for the buzzword. What is homogeneous about big data, though, is an explosion of frameworks, products and architectures that have been deployed over the last decade claiming to master big data in one way or another. These include software

[8] `https://gartner.com`

platforms such as Apache Flink[9], Apache Hadoop[10], Apache Spark[11], Apache Storm[12], IBM's InfoSphere product family[13], Google BigQuery[14], Pivotal Greenplum[15] or the lambda [129] and kappa architecture [130]. This trend suggests that one central problem behind big data indeed resides on a technological level. A critical review of current proposed solutions reveals, however, that their primary focus can be broken down to build distributed systems and many applied concepts in that direction are not new after all. They have been researched for decades right before the big data phenomenon actually evolved. Evidence for this state of affairs can be found in [131], for instance. Moreover, let us consider the lambda architecture that is often proclaimed the reference model for big data solutions. It consists of the interplay between a "batch layer" and a "real-time layer", which in turn are implemented by different distributed systems in order to combine the individual strengths of those systems (see Section 2.3.2). From these perspectives, big data should be rather considered the renaissance of distributed computing enhanced with further functionality. In fact, many desired properties of big data systems as postulated in [129] match general design goals of distributed systems (see [132]).

With this provocative discussion that certainly oversimplifies some aspects, we take the position that much of the hype underneath big data is concerned with building scalable distributed systems that are reliable. In this context, we concretely mean "horizontal scalability" (or scale-out) in which a cluster of interconnected computer systems (or nodes) can be enriched with further hardware resources by appending more nodes to speed up performance. This is in contrast to "vertical scalability" (or scale-up) that pursues the same objective but on a local level, i.e. only a single machine is considered instead of a cluster and equipped with more "central processing units" (CPUs), "random access memory" (RAM) or "hard disk drives" (HDDs) to obtain an appropriate "speedup".

2.3.2 Reference Architectures

In the last Section 2.3.1, we briefly mentioned the lambda architecture with its main building blocks, i.e. the batch layer typically realized by batch processing systems like Hadoop or Spark and the real-time layer commonly built on top of stream processors such as Flink or Storm. This architecture expects new data to arrive from one or multiple data sources that are passed over to batch and real-time layer simultaneously. From there, data are processed according to purpose and finally made available for analytics through batch and real-time views. In that sense, both layers are a blessing and curse at the same time. On the one hand, the real-time layer is required in order to compensate the latency sensitive batch layer, which means that batch views are never up to date with respect to most recent data arriving to the architecture. To compensate these

[9] `https://flink.apache.org`
[10] `https://hadoop.apache.org`
[11] `https://spark.apache.org`
[12] `https://storm.apache.org`
[13] `https://ibm.com/analytics/information-server/`
[14] `https://cloud.google.com/bigquery/`
[15] `https://greenplum.org`

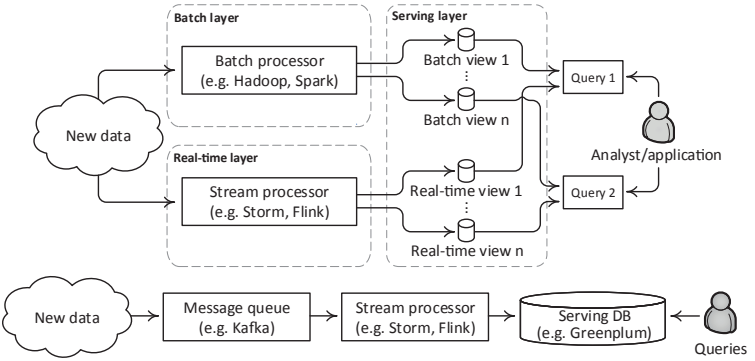

FIGURE 2.9: Common big data architectures: lambda (top) and kappa architecture (bottom) (adapted from [129, 130])

delays, the real-time layer sets in and provides these missing results in corresponding real-time views as it carries a much lower latency. Once delayed computations become available in a batch view, they are instantly removed from the real-time view to maintain consistency. As such real-time capabilities of the architecture can be met despite the mentioned drawbacks by merging the results of batch and real-time view represented in the "serving layer". With these processing principles at hand, we also have to consider higher implementation overhead on the other hand because extending functionality or other code changes certainty affect both the batch and real-time layer. According to [130], this is a serious affair in practice making the lambda architecture extremely complex from both an extensibility and maintenance standpoint. Therefore, the work supplies a counterproposal by introducing the kappa architecture. In essence, it is a simplification of the lambda architecture by getting rid of the batch layer and attempting to compensate the lost functionality, i.e. the reprocessing of historic data stored in the batch layer. To understand the compensation, we first have to look at the concept of "message-queuing" (see [132]). The main intention of message-queuing systems is to decouple processing from data producers by allocating an intermediate buffering mechanism. For that purpose, they are also frequently utilized to implement the lambda architecture in order to manage data ingestion for the batch and real-time layer. Yet, the potentials of message queues are not fully exploited in that particular context as the level of data retention is usually small. Thus, [130] suggests to expand the buffering capacity appropriately such that reprocessing demands can be enabled from that source. A well-established message-queuing system serving this idea is Apache Kafka[16] as it is horizontal scalable, fault-tolerant and able to buffer huge volumes of data. This way, data of interest can be repassed to the real-time layer or a separate instance of it in order to avoid contention issues with respect to the live workload. Hence, the kappa architecture can be a useful and lightweight alternative for several big data scenarios where it is not required to keep the entire stack of historic data. An overview of both big data architectures is depicted in Figure 2.9 shaping all central components.

[16] `https://kafka.apache.org`

2.3.3 Stream Processing Using Flink

When entering "velocity" into the equation of big data scenarios, the previous Section 2.3.2 argued that a real-time layer has to be established. This is due to the nature of input data that literally turns into a "stream", which can be defined as a potentially unbounded sequence of data. Consequently, a system has to be deployed in that layer capable to process and mine input data continuously and promptly with low latency. A natural fit for this problem are "stream processing systems". In this section, we provide a brief overview of the concrete stream processor Flink, which is a state-of-the-art representative for this sort of systems. Before we start, we want to clarify that batch processing is a special case of stream processing because batches can be considered bounded streams.

Originally evolved from a research project in 2014, Flink [133, 134] is now a top-level project of the Apache software foundation[17] with a straightforward programming model underneath to build scale-out and fault-tolerant streaming applications. It is all about building a pipeline of activities to transform and analyze input data resulting in a "directed acyclic graph", which is named "data flow" in Flink terminology or simply termed "Flink program" once compiled. Each vertex in that graph refers to an "operator" that implements custom functionality ranging from simple manipulations to window calculations and each edge reflects a stream transporting data between operators. Special operators managing data ingress and egress are "sources" and "sinks". A data flow may contain multiple of these specific operators. This means that concurrent input data sources can be consumed by subsequent operators and produced results are sent to sinks emitting the data to any desired location including DBs, log files or even new streams justifying the application logic of a Flink program. When it comes to scalability, a fundamental characteristic of Flink is to distribute the computation of a defined data flow over a cluster of participating Flink nodes. This parallel execution is realized by "operator subtasks" and "stream partitions". In that sense, subtasks can be understood as parallel instances of a particular operator all running in separated threads and stream partitions are the corresponding inputs and outputs of subtasks. As these partitions flow independently between subtasks, there may be scenarios requiring a reorganization of those substreams. For instance, let us consider that the number of parallel subtasks of one operator diverges with the parallelism of the next. There is an obvious mismatch that needs to be handled by Flink rebalancing the partitions. An exemplary data flow is depicted in Figure 2.10 where repartitioning of the substreams take place between source and the second operator. In this context, each operator instance of a Flink program can be run on different computing node or on a single one dependent on the configuration. This guides us to the anatomy of a Flink cluster. Such a cluster follows a "master/-worker pattern" where the work, that can be multiple data flows issued at the master, is scheduled and distributed among the workers. In this respect, each worker holds at least one "task slot" executing one or multiple subtasks. The latter, referred to as "slot sharing", is the default mode. It is beneficial to reduce network "input/output" (i/o) because entire processing pipelines may be executed in a single slot. The downside of slot sharing is that subtasks have to share main memory reserved per slot. The other extreme occurs when a cluster is configured to host only one particular subtask per slot.

[17] `https://apache.org`

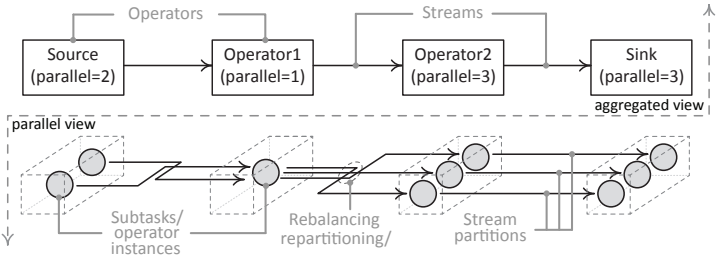

FIGURE 2.10: Simple Flink data flow from different perspectives (adapted from [135])

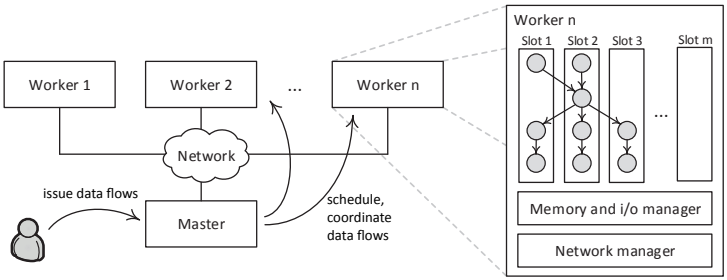

FIGURE 2.11: Anatomy of a Flink cluster with one issued program executed at a dedicated worker using slot sharing (adapted from [135])

Even though it permits to manage and control the exact amount of available memory for intensive calculations, such a setting often leads to situations where some slots carry out the heavy lifting and others essentially do very little. This imbalance binds unused resources as all, let say $m \in \mathbb{N}$, slots per worker have an equal proportion of $1/m$ of all available managed memory at their disposal to work with. A conceptual picture of a Flink cluster is outlined in Figure 2.11 with $m > 3$. This way, the simple Flink program of Figure 2.10 can run on a single worker where slot sharing is enabled.

With this basic overview, there is much more to be explored about Flink's capabilities including the realization of operator specifics and fault tolerance among others which we partially cover in Chapter 10. Anyway, further reading can be found in [133, 134] and in Flink's official documentation [135].

2.3.4 Parallel Relational Databases

Relational DB systems have a long success story that started in the late 1960s and early 1970s with the seminal works of E. F. Codd organizing data along tables, i.e. the "relational model" (see [136, 137]). Due to their maturity level reached up until today, such systems can be considered as de facto standard to implement data management aspects in enterprises and institutions. These include the handling of transactions, concurrency, storage organization and efficient data retrieval operations among others. In

light of increasing demands as given by big data scenarios where such data management characteristics are frequently desired, relational DB systems must scale with the underlying problem domain. This can be achieved by shifting away from legacy systems with uniprocessor support towards "massive parallel processing". In this section, we present basic concepts that attempt to deliver a fundamental fit to this challenge.

Originally elaborated in [138], a skeleton for the categorization of parallel DB implementations can be furnished along three distinguishing architectures, i.e. (i) "shared-everything" (initially called "shared-memory"), (ii) "shared-disk" and (iii) "shared-nothing". The most basic form is provided by systems falling into category (i) in which CPUs share the same data usually residing on HDDs that in turn are accessible via the buffer pool located in main memory. This setup permits a fairly easy realization because transaction and buffer management are directly accessible to all processing units via the main memory. As this shared-everything concept is very generic, DB manufacturers could simply port their legacy DB solutions to multiple CPUs in the past. From this evolutionary perspective, one can acknowledge that all standard commercial products follow this architecture nowadays [139]. This includes versions of IBM's Db2[18], Microsoft's SQL Server[19] and Oracle's DB[20]. Besides simplicity, a downside of the shared-everything scheme is its moderate scalability. Even though, we can increase parallel processing capabilities by appending more CPUs and RAM to such a system (see vertical scalability), the amount of addable computing resources is finite in case of bare-metal hardware. Another problem is that any hardware defect causes the overall system to go down, i.e. a "single point of failure" (SPoF). Both issues are addressed by the other two concepts that incorporate a looser coupling of the hardware components reached by a higher level of distribution. In systems of category (ii), independent nodes with processor and dedicated RAM share a common data repository via a shared-storage bus, which typically involves a "storage area network" (SAN) or a "network attached storage" (NAS). The communication between nodes is realized by a conventional network, i.e. the "interconnect". In comparison to the shared-everything scheme, this architecture is very appealing as we can get over the scaling limits by simply adding more nodes to the overall system cluster (see horizontal scalability), while maintaining a centralized view on the data. A further benefit is that a node failure does not provoke an entire cluster outage because other nodes can take over. Yet, the problem with systems of that kind is that each node holds a private buffer pool and so coherence need to be secured among distributed nodes for cached data. Certainly, this adds additional complexity (e.g. [139, 140]). It should also be stated that due to this cache coherence problem and other communication among the nodes, a high-speed network is recommended for the interconnect. A similar argumentation holds for the shared-storage bus. Hence, an appropriate sizing is indispensable to eradicate network i/o bottlenecks and to avoid disk contention. Prominent implementations of the shared-disk architecture are IBM's Db2 for zSeries[21] and Oracle's Real Application Clusters[22] (RACs) (see [139]). The last category (iii) is the most advanced in terms of decoupling. Basically, it is very similar to the shared-disk setup but data is not shared physically, i.e. each node holds a local

[18] `https://ibm.com/products/db2-database/`
[19] `https://microsoft.com/sql-server/`
[20] `https://oracle.com/database/`
[21] `https://ibm.com/analytics/db2/zos/`
[22] `https://oracle.com/database/technologies/rac.html`

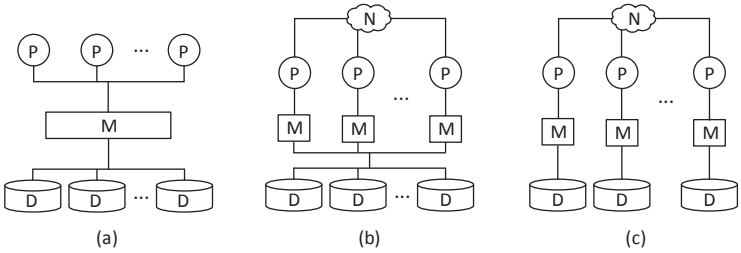

FIGURE 2.12: The three basic parallel DB architectures: (a) shared-everything, (b) shared-disk and (c) shared-nothing consisting of processors (P), memory (M), disk storage (D) and network interconnect (N) (adapted from [141])

storage. Horizontal scaling is achievable intuitively by making available a full copy of the data to each node in the cluster. More common though is a "horizontal partitioning" (or sharding) of the data with respect to specific strategies. These include storing data based on the round-robin principle, attribute ranges or hashing such that records within the same range or with the same hash value are associated with the same cluster node. According to the authors in [139], this feature is very characteristic as opposed to the other described architectures. However, the same authors admit that partitioning is a crucial factor for the efficiency of the cluster and puts a significant burden to administrative stuff as skewed data obviously prevents proper balancing of workloads among the nodes and constraining distributed query executions. If skew is unavoidable, a high-speed interconnect is a potential remedy. The appropriate sizing of the interconnect is also valuable to reduce bottlenecks due to involved coordination activities introduced by the distributed buffer and transaction management. On account of the intended local autonomy of shared-nothing solutions, fairly high availability guarantees can be realized if partition replicates are introduced in addition that minimize SPoFs. All of these points demonstrate that this architecture certainly introduce a much higher complexity compared to systems falling in category (i) and (ii) [140]. Despite this fact, it should be noted that shared-nothing clusters are gaining momentum, which can be verified along a very active development community in recent years. Exemplary projects are Pivotal Greenplum or Postgres-XL[23]. Concluding the discussion, Figure 2.12 sketches all three presented parallel DB architectures.

[23] `https://postgres-xl.org`

Part II

Uncertainty Management Aspects

Chapter 3

Rudiments of Rough Set Theory

Even though uncertainty is a multidimensional phenomenon with various complex facets to look at, its impact to a learning problem can be understood intuitively as the classification performance of a model, suffering under uncertainty, can degrade severely. An established mathematical framework to analyze data under uncertainty is provided by RST. Its principles and resulting approximations are of fundamental importance for various aspects in this thesis. Therefore, we give a brief introduction to this theory herein. After outlining some basic motives for the cause of uncertainty, underlying data structures of RST are pictured followed by its formal perception to envision information in a granular way. On these grounds, several definitions are specified to inspect the data of interest. With these approximations in place, we are also able to assess dependencies and distinguish essential from dispensable information, which is very important when it comes to solve classification problems. Besides these notions that are at the heart of classic RST, we also introduce a generalization of this theory incorporating variable precision. This way, minor irregularities in the data can be compensated by tolerating a controlled degree of uncertainty, which can be extremely valuable in practice particularly in such situations where data tends to be imperfect.

3.1 Introduction

The deeper meaning of the term "uncertainty" is versatile and generally a matter of philosophical debates. Yet, we perhaps may break down its inherent complexity to an intuitive grasp. If knowledge is understood as the essence of cognition and/or a learning process from available information, then uncertainty virtually arises from limited knowledge as we are unable to explain specific situations properly: we are simply uncertain. By way of analogy to this perception, uncertainty can be ported to ML straightforwardly where a trained learner represents some sort of knowledge. As a consequence, uncertainty may cause a degradation of predictive performances at the very end when not handled appropriately. In fact, we already experienced a notion of uncertainty implicitly in the context of learning in nonstationary environments where the central assumption of ML vanishes (see Section 2.2.3). Under such circumstances, earlier acquired knowledge becomes outdated and is no longer sufficient to solve a problem reliably because the task itself has shifted partially or completely. This lack of knowledge to describe the changed or new situation obviously generates uncertainty and requires more sophisticated learning techniques. Another form of uncertainty is not directly related to a fundamental change of the underlying data distribution but may originate from the presence of information deficiencies posing similar effects ultimately. These include information that is fragmentary, incomplete, imprecise or vague (see [142] for a complete discussion). It is not difficult to acknowledge that such imperfections in the data are quite common for real-world problems, which clearly constitutes another hurdle to build up proper knowledge. For a moment, let us consider the problem of network monitoring where NetFlow/IPFIX data are available exclusively (see Section 2.1.1). The abstracted view produced by flows comprises a coarser granularity compared to native packet information where huge portions of original network data are essentially lost. This information loss renders uncertainty quite naturally and requires sophisticated remedial action. In this chapter, we deal with the quantification of uncertainty caused by information deficiencies and particularly with those that emerge from vagueness using RST [143, 144]. This theory was developed by Z. Pawlak and his fellow researchers in the early 1980s and is a profound mathematical framework for data analysis under uncertainty. In contrast to other established theories like "fuzzy set theory" [145] or "evidence theory" [146] that can handle uncertainty as well, the main advantage of RST, in its basic formulation, is that it only relies on information that is available. It does not require meta data or other supplemental parameters such as a degree of membership or probability assignments (e.g. [147, 148]). As a matter of fact, the only assumption made by RST is that knowledge can be immediately perceived from information by using the notion of "indiscernibility". This principle is closely connected to "Leibniz's law of indiscernibility"[1]. Loosely, it states that examples, instances or objects are indiscernible if and only if all of their characteristics (properties, attributes, features) are identical. In RST, though, indiscernibility is defined relative to a given set of characteristics. From this perspective, a set of indiscernible objects forms a "granule of knowledge" and any union of these granules is referred to be "crisp" or "exact", while any other set is termed "rough" or "vague" otherwise [148].

[1] This law is named after the German polymath G. W. Leibniz who formulated the law between the 17th and 18th century.

In the remainder of this chapter, we introduce the basic idea behind RST that is employed to solve several intrinsic challenges of this thesis. First, we describe two distinct systems that act as elementary data structure (Section 3.2) and further detail the notion of indiscernibility building the mathematical foundation to perceive knowledge (Section 3.3). In view of this atomic knowledge, data regions are exposed to approximate target concepts (Section 3.4). Moreover, they can be leveraged to find specific attributes that are essential for classification tasks (Section 3.5). Based on this framework, an extension to classic RST is presented that incorporates variable precision (Section 3.6). Lastly, we close this chapter with a summary (Section 3.7).

3.2 Information and Decision Systems

Information can be represented in different ways. One very basic and convenient form is to structure information using spreadsheet-like formats with rows and columns. In such a table, an individual row refers to an example or object that has been observed whereas corresponding column values reflect their characteristical properties. RST also employs such a two-dimensional data structure named "information system" (IS), which is introduced by Definition 3.1.

Definition 3.1: *Given the universe of objects* $U = \{x_1, ..., x_n\}, n \in \mathbb{N}$ *and the attributes or features* $A = \{a_1, ..., a_m\}, m \in \mathbb{N}$*, the pair*

$$\langle U, A \rangle \tag{3.1}$$

constitutes an IS where each $a \in A$ *describes individual characteristics of* $x \in U$ *given by* $a : U \to V_a$ *with* a*'s value range* V_a*. Thus,* $a(x) \in V_a$ *specifies the concrete value of object* x *with respect to attribute* a*.*

While in many problem domains it is sufficient to shape data by an IS, we often have to face some context-specific decision made by an expert or teacher in addition. For such situations, RST provides an extension to ISs, i.e. a "decision system" (DS). Formally, it can be described by Definition 3.2.

Definition 3.2: *Given the universe of objects* $U = \{x_1, ..., x_n\}, n \in \mathbb{N}$*, the condition attributes* $A = \{a_1, ..., a_m\}, m \in \mathbb{N}$ *and decision attributes* $D = \{d_1, ..., d_p\}, p \in \mathbb{N}$*, the tuple*

$$\langle U, A, D \rangle \tag{3.2}$$

constitutes a DS, which is similar to the notion of an IS. In addition, each $d \in D$ *describes decision-specific characteristics given by* $d : U \to V_d$ *with* d*'s value range* V_d *and* $A \cap D = \emptyset$*.*

To illustrate these definitions, let us consider an exemplary DS $\langle U, A, D \rangle$ with $U = \{x_1, ..., x_{10}\}, A = \{shape, size, color\}$ and $D = \{d\}$ given by Table 3.1. With the value ranges $V_{shape} = \{$'circle','tringale'$\}$, $V_{size} = \{$'small','large'$\}$, $V_{color} = \{$'white','gray'$\}$ and $V_d = \{0, 1\}$, we have $size(x_2) = size(x_9) =$ 'large' and $d(x_2) \neq d(x_9)$, for instance. If the decision context is ignored setting $D = \emptyset$, we obtain an IS quite naturally.

-	shape	size	color	d
x_1	circle	small	white	1
x_2	triangle	large	gray	0
x_3	triangle	small	white	1
x_4	circle	small	gray	0
x_5	triangle	small	white	0
x_6	triangle	small	white	1
x_7	triangle	large	gray	0
x_8	circle	large	gray	0
x_9	circle	large	white	1
x_{10}	circle	small	white	1

TABLE 3.1: Exemplary DS with three condition attributes and one decision feature

By introducing these two fundamental data structures, we followed classic RST notations to which we stick throughout this chapter. For the sake of completeness, however, we also want to emphasize that the content of a DS can be expressed by alternative forms such as given by an $(m+p)$-dimensional vector $x := \langle a_1(x), ..., a_m(x), d_1(x), ..., d_p(x)\rangle$. This is very close to the input space $\mathcal{V}$ and concept space $\mathcal{C}$ outlined in Section 2.2.1 when considering $x \in \mathcal{V} \times \mathcal{C}$ as train or test example with $\mathcal{V} = V_{a_1} \times ... \times V_{a_m}$ and $\mathcal{C} = V_{d_1} \times ... \times V_{d_p}$. In terms of DB terminology, a further elaboration is furnished in the next Chapter 4.

3.3 Indiscernibility Relation

Given those data organization aspects, let us focus on the notion of indiscernibility. As briefly highlighted earlier, it is at the heart of RST to distinguish objects and to turn information into knowledge ultimately. In its basic form, indiscernibility can be introduced by a binary relation provided through Definition 3.3.

DEFINITION 3.3: *Let be an IS $\langle U, A\rangle$ with $B \subseteq A$, then*

$$IND(B) = \{\langle x, y\rangle \in U^2 \mid a(x) = a(y), \forall a \in B\} \tag{3.3}$$

defines the "indiscernibility relation".

Based on this definition, we can infer three essential properties, namely, reflexivity (i.e. $\forall x \in U : \langle x, x\rangle \in IND(B)$), symmetry (i.e. $\forall x, y \in U : \langle x, y\rangle \in IND(B) \Rightarrow \langle y, x\rangle \in IND(B)$) and transitivity (i.e. $\forall x, y, z \in U : \langle x, y\rangle \in IND(B) \wedge \langle y, z\rangle \in IND(B) \Rightarrow \langle x, z\rangle \in IND(B)$) forming an "equivalence relation" eventually. As to any equivalence relation, $IND(B)$ induces a partition $U/IND(B)$ over observations U and information in B with pairwise disjoint equivalence classes K_j denoted by $U/IND(B) = \{K_1, ..., K_q\}, 1 \leq j \leq q \in \mathbb{N}$. For short, we also write U/B to indicate the resulting partition.

According to Pawlak's vision, each of those equivalence classes, induced by a partition of the indiscernibility relation, corresponds to a granule of knowledge as all objects contained in such a class are indistinguishable. To highlight the granulation capabilities of this binary relation, let us revert to the DS $\langle\{x_1, ..., x_{10}\}, \{shape, size, color\}, \{d\}\rangle$ illus-

trated in Table 3.1. For instance assessing the attribute sets $B_1 = \{shape, size, color\}$, $B_2 = \{shape, size\}$ and $B_3 = \{shape\}$ in isolation, three different partitions are inducible straightaway:

$$U/B_1 = \{\underbrace{\{x_1, x_{10}\}}_{K_1}, \underbrace{\{x_2, x_7\}}_{K_2}, \underbrace{\{x_3, x_5, x_6\}}_{K_3}, \underbrace{\{x_4\}}_{K_4}, \underbrace{\{x_8\}}_{K_5}, \underbrace{\{x_9\}}_{K_6}\} \tag{3.4}$$

$$U/B_2 = \{\underbrace{\{x_1, x_4, x_{10}\}}_{K_1}, \underbrace{\{x_2, x_7\}}_{K_2}, \underbrace{\{x_3, x_5, x_6\}}_{K_3}, \underbrace{\{x_8, x_9\}}_{K_4}\} \tag{3.5}$$

$$U/B_3 = \{\underbrace{\{x_1, x_4, x_8, x_9, x_{10}\}}_{K_1}, \underbrace{\{x_2, x_3, x_5, x_6, x_7\}}_{K_2}\} \tag{3.6}$$

Considering information comprised by B_1, object x_1 and x_{10} are indiscernible because both share the same attribute values. Consequently, they belong to same equivalence class. With fewer information, exemplified by B_2, the partition induced by the indiscernible relation becomes coarser as less characteristics are at hand to describe and separate objects properly. For example, the earlier objects x_1 and x_{10} are still in the same equivalence class but this class also holds x_4, which indicates that information contained in B_2 is obviously less precise to discern x_4 from the other two objects. This is due to the removal of attribute *color* that clearly differentiated the objects beforehand. This lack of knowledge becomes even more apparent when reviewing B_3 where only the attribute *shape* is present to quantify the data. As such, the granulation is much coarser compared to B_1 and B_2 where objects fall in one out of two distinct and non-overlapping groups, i.e. objects are either circles (see K_1) or triangles (see K_2). In this respect, an analogy can be drawn to human cognition because even for us it becomes harder to distinguish and categorize objects if less details are at our disposal.

3.4 Concept Approximation

Earlier, we gave a brief introduction to build theories using supervised ML with the ambition to describe existing and unseen data as accurate as possible (see Section 2.2.1). In this section, we are eager to make a similar examination by using knowledge granules as we have seen that the indiscernibility relation can be exploited as valid tool set to classify data. Initially, however, we take a step back and try to approximate a subset of given observations only, i.e. "concept learning" in its basic form [57]. Later on, we also extend this notion to multiple concepts.

Starting with the description of a single target concept, let us reconsider the system $\langle\{x_1, ..., x_{10}\}, \{shape, size, color\}, \{d\}\rangle$ of Table 3.1. Further assuming a given target concept $X = \{x \in U \mid d(x) = 1\}$ that is to be characterized using all available information, it becomes clear that X cannot be expressed properly in terms of $U/\{shape, size, color\}$. Even though $K_1 = \{x_1, x_{10}\}$ and $K_6 = \{x_9\}$ are completely included in X, $K_3 = \{x_3, x_5, x_6\}$ is not (see (3.4)). This causes an issue because we simply cannot assert and reason why $\{x_3, x_6\} \subseteq X$ but $x_5 \notin X$ although all three objects are indistinguishable according to $\{shape, size, color\}$. This idea can be concretized in a more general form by introducing the following two Definitions 3.4 and 3.5.

Definition 3.4: *Let be an IS* $\langle U, A\rangle, B \subseteq A$ *and the target concept* $X \subseteq U$*, then*

$$\underline{X}_B = \bigcup\{K \in U/B \mid K \subseteq X\} \tag{3.7}$$

defines the "B-lower approximation" of X*.*

Definition 3.5: *Let be an IS* $\langle U, A\rangle, B \subseteq A$ *and the target concept* $X \subseteq U$*, then*

$$\overline{X}_B = \bigcup\{K \in U/B \mid K \cap X \neq \emptyset\} \tag{3.8}$$

defines the "B-upper approximation" of X*.*

In light of the B-lower and B-upper approximation, all objects in $\underline{X}_B$ can be classified with certainty to be in X, while objects in $\overline{X}_B$ are possibly in X regarding observations U and their characteristics B. Moreover, we can verify that both are crisp in terms of the indiscernibility relation, i.e. either set is the union of granules. Summarized by the pair

$$\langle \underline{X}_B, \overline{X}_B\rangle \ , \tag{3.9}$$

they are commonly known under the term "rough set" or "concept approximation". One can easily verify that $\underline{X}_B \subseteq X \subseteq \overline{X}_B$. If the B-lower and B-upper approximation are not identical, borderline examples emerge, which guides us to another approximation given by Definition 3.6.

Definition 3.6: *Let be an IS* $\langle U, A\rangle, B \subseteq A$ *and the target concept* $X \subseteq U$*, then*

$$\overline{\underline{X}}_B = \overline{X}_B \setminus \underline{X}_B \tag{3.10}$$

defines the "B-boundary approximation" of X*.*

Using the B-boundary approximation (sometimes also called "area of uncertainy" [147]), we can expose objects of equivalence classes that portray X vaguely (roughly, imprecisely) when $\overline{\underline{X}}_B \neq \emptyset$. This point essentially means that the expressiveness of B on available observations is insufficient suggesting to collect further facts in the problem domain to separate the data. Vagueness can also be captured in numeric terms given by the following coefficient

$$\theta_B(X) = \frac{|\underline{X}_B|}{|\overline{X}_B|} \ . \tag{3.11}$$

With $X \neq \emptyset$, we obtain $0 \leq \theta_B(X) \leq 1$ straightforwardly. Similar to the previous terminology, the target concept X is said to be vague ($\theta_B(X) < 1$) or exact ($\theta_B(X) = 1$) otherwise, whereas the degree of vagueness increases as $\theta_B(X)$ gravitates to 0. Illustrating these notions, let us revert our exemplary DS $\langle\{x_1, ..., x_{10}\}, A, \{d\}\rangle$ of Table 3.1 with $A = \{shape, size, color\}$ where we are given the target concept $X = \{x \in U \mid d(x) = 1\}$ to be examined. Incorporating all condition features A, we receive the following approximations

$$\underline{X}_A = \{x_1, x_9, x_{10}\}, \overline{X}_A = \{x_1, x_3, x_5, x_6, x_9, x_{10}\} \text{ and } \overline{\underline{X}}_A = \{x_3, x_5, x_6\} \ , \tag{3.12}$$

which yields

$$\theta_A(X) = \frac{|\{x_1, x_9, x_{10}\}|}{|\{x_1, x_3, x_5, x_6, x_9, x_{10}\}|} = \frac{1}{2} \ . \tag{3.13}$$

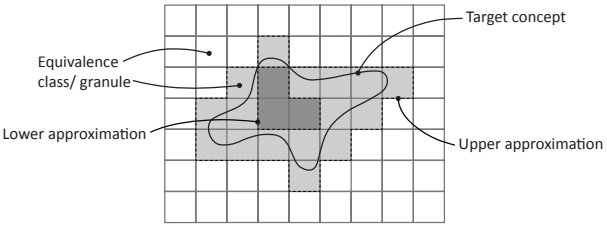

FIGURE 3.1: Prototypical illustration approximating a target concept using rough sets

This means that the approximation of X is fairly rough as half of the target concept, in fact, cannot be explained using knowledge residing in A. Obviously, this state of affairs is caused by the ambiguity of those three objects x_3, x_5 and x_6. Despite their indiscernibility, they are only partially contained in X, which ultimately renders uncertainty from a learning perspective. Independent from this assessment, a visual illustration of a prototypical rough set is outlined in Figure 3.1 where 19 granules are involved describing a given concept out of which only three are precise. The other 16 equivalence classes picture the target vaguely.

If we are eager to analyze not one but multiple target concepts $X_1, ..., X_k$ in a DS $\langle U, A, D\rangle$ modeled by decision features $E \subseteq D$, RST can also be applied to such a setting. As a matter of fact, the indiscernibility relation can be exploited to expose those k non-overlapping concepts using the resulting partition $U/E = \{X_1, ..., X_k\}, k \in \mathbb{N}$. Consequently, we can calculate each individual lower approximation guiding us to Definition 3.7.

DEFINITION 3.7: *Let be a DS* $\langle U, A, D\rangle$*,* $B \subseteq A$ *and* $E \subseteq D$*, then*

$$POS_B(E) = \bigcup_{X \in U/E} \underline{X}_B \tag{3.14}$$

defines the "B-positive region" that encompasses all objects describing the target concepts $X \in U/E$ *with certainty.*

In those cases where we are interested to outline equivalence classes explaining concepts roughly, one has just to build the complement of $POS_B(E)$. The following Definition 3.8 concretizes this undertaking.

DEFINITION 3.8: *Let be a DS* $\langle U, A, D\rangle$*,* $B \subseteq A$ *and* $E \subseteq D$*, then*

$$BND_B(E) = \bigcup_{X \in U/E} \overline{X}_B = U \setminus POS_B(E) \tag{3.15}$$

defines the "B-boundary region" that encompasses all objects describing the target concepts $X \in U/E$ *vaguely.*

With these capabilities to approximate concepts, RST can be suitable for various areas of application apparently. Aside from the exemplified quantification of vagueness, it is applicable for rule induction tasks or feature dependency analysis to name a few. Particularly for problems of the latter kind, the next Section 3.5 outlines key notions.

3.5 Reducts and Core

In many applications, the question arises whether all available condition attributes are really necessary for a particular task. Considering a classification problem, there may be features which do not contribute much while others are essential to solve the problem with high predictive performance. In such situations, one is interested to find an attribute subset with equivalent or very similar expressiveness compared to all available condition features in order to reduce the complexity of the underlying problem. The task to obtain such valuable sets of attributes is termed "feature selection" (FS). A formal perspective to this subject is provided by RST introducing the notion of "reducts" and "core". In this section, we shape and discuss the basic idea behind both terms and outline a prominent algorithm that tries to extract reducts intuitively.

We start the discussion by giving the following Definition 3.9. It states the characteristics of a reduct, which makes use of the positive region as sophisticated quality measure.

DEFINITION 3.9: *Let be a DS* $\langle U, A, D\rangle, B \subseteq A$ *and* $E \subseteq D$. *Attribute sets* $R \subseteq B$ *fulfilling the two conditions*

$$POS_R(E) = POS_B(E) \tag{3.16}$$

and

$$POS_{R\setminus\{a\}}(E) \neq POS_R(E), \forall a \in R \tag{3.17}$$

are called reduct with respect to the classification of E.

Based on (3.16), a reduct R has the same predictive quality than B on E and is minimal in the sense that no attribute can be removed without degrading this quality (see (3.17)). Attributes fulfilling this irreducible property are also sometimes referred to as "E-indispensable" or "orthogonal" in literature (e.g. [144, 149]). Dependent on the data, it is not difficult to acknowledge that there is no single reduct in place necessarily but multiple different attribute subsets can be found in a DS that comply with Definition 3.9. Consequently, this coexistence can be expressed by

$$\mathcal{R} = \{R \in \mathcal{P}(B) \mid R \text{ is a reduct}\}\,, \tag{3.18}$$

which is the family of all reducts within a given DS. Additionally, the cardinality of different reducts must not be consistent. In fact, some reducts might occur with a lower attribute cardinality than others and seeking for those is obviously a rewarding FS output. However, it has been shown that the task to find such reducts with minimal cardinality is a NP-hard problem [150]. Nonetheless, it is often sufficient for many applications to find any reduct among those available in $\mathcal{R}$ but even seeking for those can be expensive too. This can be illustrated along the following. Given a DS $\langle U, A, D\rangle$ with the condition features $B \subseteq A$ to reduce, the potential candidate subsets to inspect for the reduct properties are $\mathcal{P}(B)$, which is exemplified in Figure 3.2 showing the search space induced by three and four condition attributes. Consequently, the size of the space is equivalent to $|\mathcal{P}(B)| = 2^{|B|}$. With this exponential growth, roaming the entire search space to find reducts is prohibitive expensive even for a moderate number of condition features. Therefore, a rather systematic traversal through that space is desirable. We

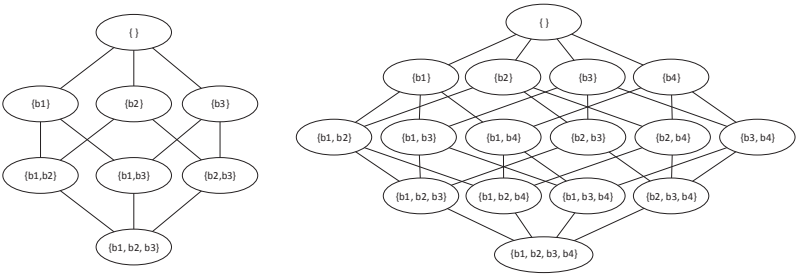

FIGURE 3.2: Hasse diagrams of two search spaces showcasing the potential feature subset candidates induced by three (left) and four condition attributes (right)

present such an algorithm at the end of this section. Before that, however, let us introduce the notion of core attributes that can be of immense benefit in practice. This set of features is directly connected to reductsand can be defined by the following Definition 3.10.

DEFINITION 3.10: *Let be a DS* $\langle U, A, D\rangle, B \subseteq A$ *and* $E \subseteq D$*. The attribute set* $C \subseteq B$ *defined by*

$$C = \{a \in B \mid POS_{B\setminus\{a\}}(E) \neq POS_B(E)\}\,, \tag{3.19}$$

is called core with respect to the classification of E*, i.e. the set consisting of all* E*-indispensable attributes.*

These core attributes $a \in C$ are extremely valuable because they cannot be removed without loosing expressive power. In that sense, the core holds the most essential information of all considered condition attributes in B for the classification of E. Particularly, one can verify the context between core and reducts given by

$$C = \bigcap_{R\in\mathcal{R}} R\,, \tag{3.20}$$

which is an alternative description to Definition 3.10 (e.g. [144, 151]). It states that the intersection of all attribute sets compliant with Definition 3.9 can also be utilized to expose the core. While (3.20) is certainly a more illustrative version of the core C, it has some practical pitfalls. The reason behind this statement is due to the fact that all reducts must be known right before we can get C. This includes even those reducts having minimal cardinality, if any, whose search is a very difficult undertaking in the general case as pointed out earlier. Therefore, we stick to Definition 3.10 if we are interested to find core attributes explicitly because (3.19) is way more attractive from a computational perspective. This is experienced during the course of this work. Lastly, let us briefly present one prominent approach that tries to seek for reducts effectively by traversing the search space instead of inspecting each and every possible attribute subset. It is called QUICKREDUCT (e.g. [151, 152]) and starts off the search with $R = \emptyset$ as initial reduct candidate, i.e. a "forward selection" process (see [99]). In each iteration, it adds a new attribute $a \in B \setminus R$ to R if that new feature subset $R \cup \{a\}$ has a higher predictive capability than previously seen attribute combinations. This is done by exploiting the positive region as heuristic function measuring the performance of the

Input: $\langle U, A, D\rangle, B \subseteq A, E \subseteq D$
Output: $R \in \mathcal{P}(B)$

```
1: BEGIN
2:   R ← ∅
3:   LOOP
4:     S ← R
5:     FOR a ∈ B \ R LOOP
6:       IF |POS_{R∪{a}}(E)| > |POS_S(E)| THEN
7:         S ← R ∪ {a}
8:       END IF
9:     END LOOP
10:    R ← S
11:    EXIT WHEN |POS_R(E)| = |POS_B(E)|
12:  END LOOP
13: END
```

Algorithm 3.1: Procedure QuickReduct to extract a single reduct

current feature subset candidate, which becomes feasible using its inherent monotone relation, i.e.

$$R \subseteq B \Rightarrow |POS_R(E)| \leq |POS_B(E)| \,, \tag{3.21}$$

whose justification, for example, can be found in [151]. Consequently, QuickReduct proceeds with this stepwise approach until R reaches the same expressiveness than B. The entire processing of QuickReduct is outlined in Algorithm 3.1. Alternatively, the feature space can also be traversed from the other direction starting with $R = B$ as initial reduct candidate such that in each iteration least informative features are removed from R. Algorithms pursuing this type of search strategy are referred to as "backward elimination" approaches (see [99]). Such an attempt is made by QuickReduct II [151], for instance.

3.6 Variable Precision

Even though RST provides a solid framework to assess data under uncertainty, it is stiff particularly when data tend to be noisy. Immediately, this can be envisioned when only one or very few objects of a given equivalence class are not related to a single target concept but for the vast majority of objects in that class it is indeed the case. Hence, it can be valuable to accept some level of uncertainty to get a better understanding of the data. This section introduces "variable precision rough sets" (VPRSs) originally coined by W. Ziarko in [153]. It is a generalization of classic RST tolerating minor irregularities in the data. Initially, we outline the notion of "majority inclusion", which is central to VPRSs (Section 3.6.1) and reorganize the concept approximation (Section 3.6.2).

3.6.1 Majority Inclusion

In RST, we approximate concepts based on equivalence classes and the standard subset inclusion. The latter instrument is relaxed when applying VPRSs and replaced by the majority inclusion operation. As such, a controlled degree of overlapping is permitted

in order to address minor irregularities in the data that would fall out or be considered inconsistent in the classic rough set model. Before heading to the majority inclusion, let us introduce its preliminary expressing the error when overlapping two conventional sets, which is given in the following Definition 3.11

DEFINITION 3.11: *Let X and Y be two subsets defined on universe U, then*

$$z(X,Y) = \begin{cases} 1 - \frac{|X \cap Y|}{|X|} & \text{, if } X \neq \emptyset \\ 0 & \text{, otherwise} \end{cases} \tag{3.22}$$

describes the "relative degree of misclassification" of X with respect to Y.

We obtain $0 \leq z(X,Y) \leq 1$. Assuming both sets X and Y to be non-empty, they are obviously disjoint in the extreme case $z(X,Y) = 1$ whereas $z(X,Y) = 0$ signals no misclassification error inferring that X is equal to Y or at least a subset of it. Permitting a controlled degree of misclassification $0 \leq \beta < 0.5$, we can guarantee that the majority of objects in X is indeed included in Y. This line of thought is formulated by Definition 3.12 and yields the conception of the majority inclusion.

DEFINITION 3.12: *Let X and Y be two subsets defined on universe U and the tolerated error $\beta \in [0, 0.5)$, then*

$$X \subseteq_\beta Y \Leftrightarrow z(X,Y) \leq \beta \tag{3.23}$$

defines the majority inclusion.

Based on this definition, it can be inferred that in those cases where no misclassification error is permitted, we have $X \subseteq_{\beta=0} Y \Leftrightarrow X \subseteq Y$, i.e. the majority inclusion corresponds to the standard subset inclusion. If we get back to our running example considering the DS $\langle \{x_1, ..., x_{10}\}, A, \{d\} \rangle$ of Table 3.1 using $A = \{shape, size, color\}$ with the given target concept $X = \{x_1, x_3, x_6, x_9, x_{10}\}$, we can easily figure out that for equivalence class $K_3 = \{x_3, x_5, x_6\} \in U/A$ (see (3.4)), it is not true that $K_3 \subseteq X$ but $K_3 \subseteq_\beta X$ when admitting the error $1/3 \leq \beta < 1/2$.

3.6.2 Beta Approximation

Using the notion of the majority inclusion as introduced in the previous Section 3.6.1, we are able to elevate the traditional rough set definitions (see Section 3.4) to compensate minor irregularities in the data controlled by the parameter β. Definition 3.13 starts off with the redefinition of the B-lower approximation.

DEFINITION 3.13: *Let be an IS $\langle U, A \rangle, B \subseteq A$, the target concept $X \subseteq U$ and the tolerated error $\beta \in [0, 0.5)$, then*

$$\underline{X}_B^\beta = \bigcup \{K \in U/B \mid z(K,X) \leq \beta\} \tag{3.24}$$

defines the "β-lower approximation" of X.

Hence, we directly can catch the parallels to the classic B-lower approximation (see Definition 3.4) because $\underline{X}_B^\beta$ only contains those equivalence classes $K \in U/B$ where

$K \subseteq_\beta X$ holds true. On the contrary, the β-upper approximation is given by Definition 3.14 .

Definition 3.14: *Let be an IS* $\langle U, A\rangle, B \subseteq A$, *the target concept* $X \subseteq U$ *and the tolerated error* $\beta \in [0, 0.5)$, *then*

$$\overline{X}_B^\beta = \bigcup\{K \in U/B \mid z(K, X) < 1 - \beta\} \tag{3.25}$$

defines the "β*-upper approximation" of* X.

We refer to both crisp sets as "β-approximation". Yet, there might be still borderline examples despite the introduced tolerance. These can be exposed using Definition 3.15.

Definition 3.15: *Let be an IS* $\langle U, A\rangle, B \subseteq A$, *the target concept* $X \subseteq U$ *and the tolerated error* $\beta \in [0, 0.5)$, *then*

$$\underline{X}_B^\beta = \bigcup\{K \in U/B \mid \beta < z(K, X) < 1 - \beta\} \tag{3.26}$$

defines the "β*-boundary approximation" of* X.

An alternative description of the β-boundary approximation can be deduced when reviewing (3.26) from a set-theoretic perspective as we are interested in those classes K fulfilling $z(K, X) < 1 - \beta$ (see (3.25)) except for those K with $z(K, X) \leq \beta$ (see (3.24)). As such, $\underline{X}_B^\beta = \overline{X}_B^\beta \setminus \underline{X}_B^\beta$ follows straightaway, which basically corresponds to the traditional Definition 3.6.

By analogy to classic rough sets, we are oftentimes interested to approximate multiple target concepts instead of a single one. Consequently, VPRSs provide further definitions to address problems in this direction. We refer to them as "β-regions" given by the following three Definitions 3.16, 3.17 and 3.18.

Definition 3.16: *Let be a DS* $\langle U, A, D\rangle$, $B \subseteq A, E \subseteq D$ *and the tolerated error* $\beta \in [0, 0.5)$, *then*

$$POS_B^\beta(E) = \bigcup_{X \in U/E} \underline{X}_B^\beta \tag{3.27}$$

defines the "β*-positive region" that encompasses all objects describing the target concepts* $X \in U/E$ *with certainty.*

Definition 3.17: *Let be a DS* $\langle U, A, D\rangle$, $B \subseteq A, E \subseteq D$ *and the tolerated error* $\beta \in [0, 0.5)$, *then*

$$BND_B^\beta(E) = \bigcup_{X \in U/E} \overline{X}_B^\beta \,. \tag{3.28}$$

defines the "β*-boundary region" that encompasses all objects describing the target concepts* $X \in U/E$ *vaguely.*

Definition 3.18: *Let be a DS* $\langle U, A, D\rangle$, $B \subseteq A, E \subseteq D$ *and the tolerated error* $\beta \in [0, 0.5)$, *then*

$$UPP_B^\beta(E) = \bigcup_{X \in U/E} \overline{X}_B^\beta \tag{3.29}$$

defines the "β*-upper region" that encompasses all objects describing the target concepts* $X \in U/E$ *possibly.*

Lastly, we want to reemphasize that the notions comprised by VPRSs are a generalization of classic RST. In this respect, one can verify that in case of $\beta = 0$ both models are equivalent.

3.7 Summary

Uncertainty is a huge challenge when it comes to the design and develop of learning systems. Virtually, it arises from a lack of knowledge that may have different origins but with the same undesirable effect ultimately, i.e. the learning system in charge is incapable to describe a specific situation. Hence, it is very important to quantify uncertainty and to manage it throughout the learning cycle. In this chapter, we put emphasis on uncertainty caused by information artifacts (in particular vagueness and ambiguity) and introduced RST as established formal framework to handle these deficiencies. Even though RST cannot resolve those imperfections directly, it supplies a tool set to identify vagueness and thereby stating which regions in the data can be classified roughly and which can be explained with certainty. Extracting this hidden knowledge is achieved using the notion of indiscernibility, which can be adjusted relative to the set of available information. This way, we are not only capable to look at the data from different angles with different granularities but we are also in the position to distinguish highly expressive attributes from dispensible ones to describe a given problem with most essential information only. Yet, a drawback of RST is its stiffness especially when data under inspection tend to be noisy. Under such circumstances, the vast majority of objects would be determined to be rough. Hence, it can be valuable to accept a degree of uncertainty in order to get a better understanding of the actual data and to extract hidden structures despite of minor irregularities in the data. Several extensions exist to solve such frequent situations in practice. In this chapter, we presented the VPRS model, which is one of the earliest and most established rough set generalizations found in literature. Basically, it relies on the conception of majority inclusion being much more flexible than the standard subset inclusion employed in classic RST and so it is better suited when data analysis has to be performed under noise. With these rudiments in place, we deal with RST from a practical angle in the next Chapter 4. Concretely, we devise a novel implementation that permits to compute rough sets as well as VPRSs efficiently inside relational DB systems. It is used to tackle several technical challenges in the remainder of this work.

Lastly, it should be stressed at this point that due to the rich history of RST dating back almost four decades, we only touched on the very basics of that theory in this chapter. To study deeper insights, extensions and applications, interested readers are relegated to [144, 148, 154] in order to get a starting point for further reading.

Chapter 4

In-Database Variable Precision Rough Sets

Knowledge discovery "in" DBs rather than "from" DBs has become increasingly attractive over the last decades because the view on DB systems can be stretched from transactional data storages to advanced analytical platforms. This perception has practical benefits as DB engines provide scalable algorithms for data processing. Additionally, analytics are brought close to the data such that data movements can be eliminated to a great extent. Employing RST is of particularly interest for this subject because it combines the ability to unlock hidden knowledge from data while being founded on set-theoretic operations at the same time. These operations in turn are implemented efficiently in most relational DB systems and have been constantly optimized for over four decades. Driven by the idea to utilize rough set methodology for several challenges during the course of this thesis enclosed within a solid implementation, we propose a VPRS model for relational DB engines in this chapter. For this purpose, we restructure the concept approximation to rewritten set-oriented expressions and show that these are semantically compliant in relation to the original definitions of VPRSs. This way, they can be translated to "relational algebra" easily and immediately serve common knowledge discovery tasks such as dimensionality reduction or decision rule mining, which are applicable inside conventional parallel or completely distributed DB systems right away. Our experiments indicate that the proposed in-DB model scales and it is comparable to closely related methods in terms of runtime performance but superior when facing uncertainty.

This chapter is based on:

F. Beer, U. Bühler: "An in-database rough set toolkit". In: R. Bergmann, S. Görg, G. Müller (eds.) *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB (LWA'15)*, pp. 146–157, CEUR-WS.org, 2015.

F. Beer, U. Bühler: "In-database feature selection using rough set theory". In: J. P. Carvalho, M. J. Lesot, U. Kaymak, S. Vieira, B. Bouchon-Meunier, R. R. Yager (eds.) *Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2016)*, Communications in Computer and Information Science, vol. 611, pp. 393–407, Springer, 2016.

4.1 Introduction

Mining data became more pronounced with the broad availability of ML software libraries and frameworks in the 1990s and 2000s. These include prominent general-purpose solutions such as WEKA [155], KNIME [156], RapidMiner [157] or niche players like ROSETTA [158], RSES [159] and ROSE [160] known from the RST community. Over time, these systems have been constantly improved and refined from a functional perspective towards mature platforms providing a huge arsenal of mining algorithms that literally turn conventional computer systems into analytical workbenches. As such, they are an essential ingredient for "data science" nowadays. However, these ML environments are commonly separated from the problem domain and so the typical mining process involves a stage where the data of interest must be imported prior to the actual extraction of hidden knowledge. In that sense, data are accessed through flat files or external data stores. Either way, loading mechanisms for these sources certainly do a proper job for moderately sized data sets but carry a rather poor performance for large quantities of data due to inefficient file operations or long lasting data transports over the wire. Hence, these classic loading techniques become a huge concern in the long term considering the challenge to manage ever-growing data volumes (see Section 2.3). To get over these practical shortcoming of existing ML software frameworks, a decisive paradigm evolved in data science and related disciplines, which is broadly known under the term "in-DB processing" or "in-DB analytics" (e.g. [161, 162, 163, 164, 165]). In essence, this paradigm orchestrates conventional DB infrastructures towards scalable analytical workbenches by leveraging the powerful "structured query language" (SQL) to manage data and other valuable built-in functionality. This renders substantial amenities because hidden knowledge resides in relational repositories predominantly given through transactional data or warehouses. As such, in-DB analytics has the capabilities to minimize critical data movements to external mining frameworks and to reduce processing time largely once setup adequately.

Inspired by these practical advantages of in-DB processing and to overcome the shortcomings of classic RST implementations, we make an attempt to bring VPRSs to the domain of relational DBs in this chapter. Unlike other mining approaches that are difficult to model inside DBs due to their procedural structures, VPRS or RST are well suited for this subject because they are founded on set-theoretic operations and these in turn are efficiently realized by relational engines. By redefining the fundamental concept approximation to equivalent expressions that can be implemented by modern DB systems straightforwardly, we derive a new in-DB VPRS model. It is widely applicable without the use of external software logic at low communication costs. It is tolerant to minor data irregularities and, thus, beneficial for real-life scenarios. We are further motivated by the following three points: (i) it is observable that the availability of RST methods for in-DB applications is limited and (ii) those existing solutions are either inefficient, incomplete or struggle with the handling of uncertainty. (iii) The leverage effect of mature relational DB technology given through algorithms, data structures, statistics and parallelism tightly paired with RST supply a solid match for several challenges in this thesis. Concretely, these touch scalability aspects of the architectural design, an in-depth feature analysis and the continuous extraction of certain and uncertain decision rules for the final HFIDS to develop.

To get an overview of the chapter's structure, we briefly highlight the main sections in what follows. First of all, an outline is provided on existing literature fusing RST and DBs (Section 4.2). Second, the notion of ISs and DSs are formulated in terms of DBs terminology along with important algebraic expressions (Section 4.3). Based on that, we describe our new in-DB VPRS model from a formal perspective by taking into account its relation to classic rough sets, its complexity and its implementation using SQL (Section 4.4). Moreover, we clarify the benefits of our model against some closely related approaches and state important expressions to find core and reduct attributes for FS tasks (Section 4.5). At the end of this chapter, a comparative study is conducted with respect to functionality and performance aspects (Section 4.6) followed by a discussion and a summary (Section 4.7).

4.2 Combining Rough Sets and Databases

The very beginnings of combining RST and DBs date back to the late 1980s and early 1990s. Among one of the first research works is DBRough [166]. The purpose of this system is to serve data mining tasks including dimensionality reduction and rule learning as an extension of the DBLearn system [167]. Unfortunately, concrete implementation details are omitted for both duties such that it remains unclear which relational operations are guiding the processing. Similar intentions fusing RST and DBs are formulated in [168, 169] but with the same issue of limited information regarding the underlying algebra largely refusing comprehension on the original research. Improvements and more insights of [168] are presented in [170]. Yet, basic operators are touched semantically. As such, DB internals need to be modified at the very end, which dismisses a general employment. This is particularly true applying [170] on proprietary DB systems. Parallel to the developments of DBLearn, another data mining system with an emphasize to RST was developed in [171]. It integrates SQL commands to pretreat and fetch relevant data from a back-end DB, which are finally processed on a row-by-row basis to compute VPRSs. Despite the point that pretreatments inside the DB are sort-based, a solid performance can be established as long as data can be compressed adequately. As such fewer data is transmitted in this conventional client-server architecture, which is also suggested in [172]. However, the compression capabilities of [171] are highly dependent on the data distribution and not on the employed algorithm and so transports of few data cannot be guaranteed in the general case causing enduring network i/o. Several years later, the work in [173, 174] emerged which we consider the succeeding model of DBLearn permitting to compute core and reduct attributes. This time, though, traditional DB operations are explicitly highlighted in the form of relational algebra. This approach is very efficient because the entire computation can be lifted by the respective DB engine without considerable data movements. A similar conceptual design is given by [175]. Yet, these full-featured in-DB algorithms neglect the utilization of the concept approximation and employ other elaborated rough set properties instead, which limit their application primarily to FS tasks only. Additionally, inconsistent data is considered noise and handled differently. While in [175] a noise measure is developed on predefined thresholds, records causing this uncertainty are simply eliminated in [173, 174]. Procedures capable to extract rules using DB technology are proposed in [176, 177]. Since they rest upon [173, 174], removing data inconsistency is still an

obligatory step. A completely different approach is taken in [178]. It calculates rough sets based on extended equivalence matrices inside DBs. Once data is successfully transformed into the matrix structure, the proposed methods apply but rely on procedural logic rather than on scalable DB operations. In [179], the concept approximation is extracted using multiset decision tables. The construction of such tables requires the execution of dynamic queries, helper structures and row-by-row updates as stated in [180] and, therefore, depends on inefficient preprocessing. The work in [181] implements α-RST in data warehouse environments. The algorithm relies on iterative processing and insert commands to determine the final classification. Details about its efficiency are omitted. Another model known under the term "rough relational DB" is introduced in [182]. It builds upon multi-valued relations designed to query data under uncertainty. Over the years, specific operations and properties of this theoretic model have been further extended. For instance, the authors in [183] try to port the rough relational data model to mature DB engines. However, details of migrating its algebra are not reported. Infobright [184] in turn is a physical DB system with a focus on fast data processing towards ad-hoc querying. This can be accomplished by a novel data retrieval strategy using compression, which is inspired by RST. Data are organized underneath the so-called "knowledge grid" that is employed to get estimated query results rather than seeking costly information from disk. One of the newer approaches is introduced in [185]. It frames a computational model for the lower, upper and boundary approximation usable in most conventional DBs. Yet, important notions for binary and multiclass classification are missing and so its applicability is limited. As this attempt is built upon classic RST, the handling of variable precision is not supported either.

This review demonstrates that many existing approaches leveraging DB technology are using procedural structures or external programming (e.g. [171, 178, 179, 181]). Just few methods fully exploit DB operations (e.g. [173, 174, 175, 176, 177]), but they are not entirely based on rough sets, which renders severe limitations in practice (see Section 4.5.1). Only [185] attempts to translate the classic concept approximation to relational algebra. The problem, though, is that it is solely defined for single concepts. From this perspective, it can be considered incomplete because both the positive and boundary region cannot be phrased straightaway. Given this research situation, the contribution of this chapter is two-fold. First, we build up a well-founded in-DB model capable to compute VPRSs for single as well as for multiple concept descriptions which we originally introduced in [186]. As it is a generalization of RST, we show how to compute traditional rough sets in addition thereby bridging the gap to our initial in-DB RST model [187] which, on the other end, partially relies on [185].

4.3 Basic Considerations and Database Notations

In order to bring VPRSs and FS functionality to the relational domain, some preliminaries have to be studied, which we address in this section. First, we explain the notions of an IS and DS within DB terminology. Second, we utilize relational algebra as lingua franca of relational DB theory to introduce several basic operators permitting to express the indiscernibility within DB semantics and to manage comparisons among equivalence classes respectively.

An IS is a data structure that naturally corresponds to a data table in relational terms. However, essential differences can be identified when focusing on their scope of application [188]. While an IS is used to discover regularities in the data in a snapshot fashion, the philosophy of a DB table is to serve as a repository for long-term data storing and efficient information querying respectively [189]. Despite of these contextual deviations, we try to relax them in order to work out their common grounds from a structural perspective, i.e. rows and columns. With this mindset, we can bring an IS or DS to the relational domain considering the following: let be a DS $\langle U, A, D\rangle$ with the universe of objects $U = \{x_1, ..., x_n\}$, the condition features $A = \{a_1, ..., a_m\}$ and the decision attributes $D = \{d_1, ..., d_p\}, n, m, p \in \mathbb{N}$, then we use the traditional notation of a $(m + p)$-ary DB relation

$$T \subseteq V_{a_1} \times ... \times V_{a_m} \times V_{d_1} \times ... \times V_{d_p} \tag{4.1}$$

where V_{a_i} and V_{d_j} are the domains of the corresponding condition attributes $a_i, 1 \leq i \leq m$ and decision features $d_j, 1 \leq j \leq p$, which conforms with the structure of a DS intuitively (see Section 3.2). Additionally, we permit multiset semantics on T as duplicated tuples can be handled by modern DB engines straightforwardly and write $T_{\langle a_1,...,a_m,d_1,...,d_p\rangle}$ to indicate T with its underlying attribute schema. Because such schemas are usually very long, we employ $T_{\langle A,D\rangle}$ as short form out of convenience. In terms of a conventional IS $\langle U, A\rangle$ where no decision feature is specified, we indicate the corresponding table by $T_{\langle a_1,...,a_m\rangle}$ or $T_{\langle A\rangle}$. To setup naming conventions for both types of DB relations, we call $T_{\langle A,D\rangle}$ a "decision table" and $T_{\langle A\rangle}$ a "data table" respectively. It is also worth mentioning that sometimes the membership of an attribute a to a data table T is ambiguous. In such situations, we explicitly write a_T to indicate a's affiliation to T.

Based on this harmonization, we introduce basic operations on such data tables using relational algebra (see [136, 137]) and some of its extensions (e.g. [140, 190, 191]). In particular, we concentrate on the "projection" (π), "selection" (σ), "grouping" ($\mathcal{G}$) and "join operation" ($\bowtie$). Let us begin with the projection. This operation permits to modify the attribute schema $\langle A\rangle$ of a data table $T_{\langle A\rangle}$ in such a way that all tuples $t \in T_{\langle A\rangle}$ are projected to a specified feature subset $B \subseteq A$ denoted by $\pi_B(T_{\langle A\rangle})$. As a results, a new data table $R_{\langle B\rangle}$ is created with essentially the same content compared to the original one but without those attributes $A \backslash B$ while ignoring emerging duplicates. A variant of this standard operation without enforcing a duplicate elimination is indicated by $\pi^+_B(T_{\langle A\rangle})$. An explicit tuple filtering operation is performed via $\sigma_\phi(T_{\langle A\rangle})$. The output of this expression is a new data table preserving the original attribute schema $\langle A\rangle$ but ignoring those tuples $t \in T_{\langle A\rangle}$ that are not fulfilling condition ϕ. Sometimes we are also interested to assemble two tables $S_{\langle W\rangle}$ and $T_{\langle V\rangle}$ such that tuples of both tables are joined by applying $\bowtie$. The condition that triggers the fusion process is $a_S = a_T, \forall a \in W \cap V$ and the resulting schema consists of all features comprised by W and V where equal attributes are shown only once. More formally, this behavior can also be denoted by

$$S_{\langle W\rangle} \bowtie T_{\langle V\rangle} := \sigma_{\wedge_{a\in W\cap V}(a_S=a_T)}(S_{\langle W\rangle} \times T_{\langle V\rangle}) \,. \tag{4.2}$$

Lastly, let us turn to the grouping operator $\mathcal{G}_{F,G,B}(T_{\langle A\rangle})$ that groups tuples in $T_{\langle A\rangle}$ according to the attributes $G \subseteq A$ and applies the set of specified aggregation functions $F = \{f_1, ..., f_r\}, r \in \mathbb{N}_0$ to each group. Especially, we are interested in determining a group's size or summing a numeric attribute comprised in the group using the aggrega-

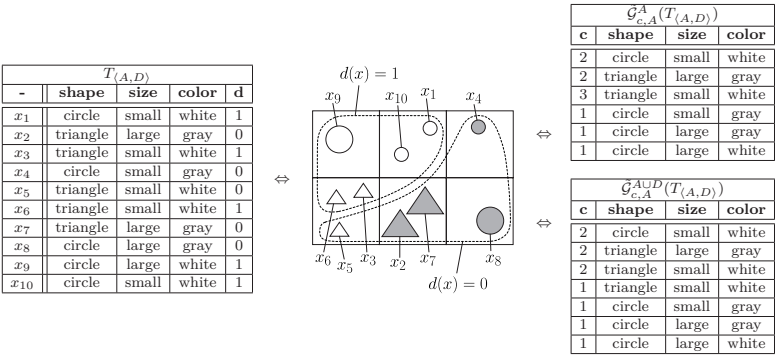

FIGURE 4.1: Mapping the indiscernibility to the relational domain: a given decision table (left) and its graphical representation (center) expressed with DB operations (right)

tion function `count` and `sum` respectively. The output schema of the resulting data table corresponds to $\langle F, B\rangle$ with $B \subseteq G$. In some sense, the behavior of $\mathcal{G}$ can be interpreted analogously to the functioning of π because $\mathcal{G}_{F,G,B}(T_{\langle A\rangle}) \Leftrightarrow \pi_B(T_{\langle A\rangle})$, for $F = \emptyset$ and $B = G$. Yet, $\mathcal{G}$ has the ability to compute statistics per group via those mentioned aggregation functions. With such a capability provided by $\mathcal{G}$, we obtain a powerful tool to mimic the indiscernibility relation based on extended relational algebra because each group in the result of $\mathcal{G}$ can be understood as an equivalence class quite naturally. Therefore, we exploit this operation as our representation of the indiscernibility given by

$$\tilde{\mathcal{G}}^{G}_{c,B}(T_{\langle A\rangle}) := \rho_{c\leftarrow \texttt{count}}(\mathcal{G}_{\{\texttt{count}\},G,B}(T_{\langle A\rangle})) \tag{4.3}$$

where $\rho_{b\leftarrow a}(R_{\langle W\rangle})$ is the renaming operator that simply changes the name of an arbitrary attribute $a \in W$ to its new name b. As such, each tuple of the expression $\tilde{\mathcal{G}}^{B}_{c,B}$, i.e. our compressed multiset representation, corresponds to an equivalence class in the induced partition U/B, while attribute c holds the cardinality of each particular class. Using a decision table $T_{\langle A,D\rangle}$, the operations of $\tilde{\mathcal{G}}$ and its parallels to the RST are depicted in Figure 4.1 with $A = \{shape, color, size\}$ and $D = \{d\}$.

4.4 Computing Variable Precision Rough Sets

Having discussed the mapping of ISs and DSs from a DB perspective alongside with basic relational operations to express the indiscernibility in Section 4.3, this section transfers VPRSs to the domain of DBs in two phases. First, we restructure the β-approximation and the β-regions to rewritten set-oriented expressions and show that these are no extensions to Ziarko's model but equivalent terms given through two propositions (Section 4.4.1). As they are very close to DB semantics, they can be ported to relational algebra intuitively. This is conducted in a second step via two theorems representing a compliant in-DB VPRS model (Section 4.4.2). Furthermore, traditional rough sets are deduced (Section 4.4.3) and details about its efficiency are outlined followed by appropriate SQL statements serving a direct deployment inside DB systems (Section 4.4.4).

4.4.1 Restructuring the Concept Approximation

Bringing Ziarko's VPRS model closer to the domain of DBs, let us start by rephrasing those single target concept definitions right before we move to a more generic variant considering all concepts induced by one or multiple decision attributes. Definition 3.13, 3.14 and 3.15 are restructured by the following Proposition 4.1.

PROPOSITION 4.1: *Let be an IS* $\langle U, A\rangle, B \subseteq A$ *and* $\beta \in [0, 0.5)$*. For any* $X \subseteq U$*, the* β*-approximation* $\underline{X}_B^\beta, \overline{X}_B^\beta$ *and* $\overline{\underline{X}}_B^\beta$ *can be rewritten to*

$$\bigcup\{K \in U/B \mid \exists H \in X/B : \phi\} \tag{4.4}$$

whereas the condition ϕ *is defined by*

$$\phi : \begin{cases} z(K,H) \leq \beta, & \text{for } \underline{X}_B^\beta \\ z(K,H) < 1-\beta, & \text{for } \overline{X}_B^\beta \\ \beta < z(K,H) < 1-\beta, & \text{for } \overline{\underline{X}}_B^\beta \,. \end{cases} \tag{4.5}$$

PROOF 4.1: *One has to compare classes* $K \in U/B$*, which have elements in* $X \subseteq U$ *and* $H \in X/B$*. We obtain* $K \cap X = H$ *for* $K \cap X \neq \emptyset$ *because of* $X \subseteq U$ *and get*

$$z(K,X) = 1 - \frac{|K \cap X|}{|K|} = 1 - \frac{|H|}{|K|} = 1 - \frac{|K \cap H|}{|K|} = z(K,H) \,. \tag{4.6}$$

It follows $z(K,X) = z(K,H) \leq \beta$*, which is proposed by* $\underline{X}_B^\beta$*. Likewise, one can show* $z(K,X) < 1-\beta$ *is equivalent to* $z(K,H) < 1-\beta$ *exposing* $\overline{X}_B^\beta$*. From those two justifications,* $\overline{\underline{X}}_B^\beta$ *can be concluded immediately since it is derivable from the set-theoretic differences* $\overline{X}_B^\beta \setminus \underline{X}_B^\beta$*.* □

Similar to the β-approximation for single concepts, Proposition 4.1 even seeks for equivalence classes $K \in U/B$ having adequate intersection with the target X bearing in mind β. The overlap, however, is not measured between K and X. Instead, X is dissected further into classes $H \in X/B$ which permits the usage of $z(K,H)$. This property is also reused next in combination with $U/(B \cup E)$ to rephrase the β-regions given through Definition 3.16, 3.17 and 3.18 via Proposition 4.2.

PROPOSITION 4.2: *Let be a DS* $\langle U, A, D\rangle, B \subseteq A, E \subseteq D$ *and* $\beta \in [0, 0.5)$*. The* β*-regions* $POS_B^\beta(E), UPP_B^\beta(E)$ *and* $BND_B^\beta(E)$ *can be rewritten to*

$$\bigcup\{K \in U/B \mid \exists H \in U/(B \cup E) : \phi\} \,, \tag{4.7}$$

whereas the condition ϕ *is defined by*

$$\phi : \begin{cases} z(K,H) \leq \beta, & \text{for } POS_B^\beta(E) \\ z(K,H) < 1-\beta, & \text{for } UPP_B^\beta(E) \\ \beta < z(K,H) < 1-\beta, & \text{for } BND_B^\beta(E) \,. \end{cases} \tag{4.8}$$

PROOF 4.2: *Using the equality* $\{H \in X/B \mid X \in U/E\} = U/(B \cup E)$*, we conclude Proposition 4.2 directly from Proposition 4.1.* □

4.4.2 Mapping Variable Precision Rough Sets

With these Propositions 4.1 and 4.2 given in the last Section 4.4.1, we are able to port VPRSs to the domain of DBs straightforwardly. First of all, we take care of a single target concept via Theorem 4.1 and introduce relational expressions to handle multiple concepts in a second stage using Theorem 4.2.

THEOREM 4.1: *Let be a data table* $T_{\langle A\rangle}, B \subseteq A, \beta \in [0, 0.5)$ *and a target concept* $X_{\langle A\rangle}$*, which is a subset of* T*. We can compute the* β*-lower* $\mathcal{L}_B^\beta(T, X)$*,* β*-upper* $\mathcal{U}_B^\beta(T, X)$ *and* β*-boundary approximation* $\mathcal{B}_B^\beta(T, X)$ *of* X *using the relational operations*

$$\pi^+_{\{c_t\}\cup B}(\sigma_\phi(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) \tag{4.9}$$

with the condition ϕ *defined by*

$$\phi : \begin{cases} 1 - \frac{c_p}{c_t} \leq \beta, & \text{for } \underline{X}_B^\beta \\ 1 - \frac{c_p}{c_t} < 1 - \beta, & \text{for } \overline{X}_B^\beta \\ \beta < 1 - \frac{c_p}{c_t} < 1 - \beta, & \text{for } \underline{\overline{X}}_B^\beta \ . \end{cases} \tag{4.10}$$

Obviously, (4.9) compares classes $K \in U/B$ with those $H \in X/B$ according to the condition attributes B and outputs those fulfilling (4.10). In this respect, $|K|$ corresponds to the aggregate c_t obtained via $\tilde{\mathcal{G}}^B_{c_t,B}(T)$ and $|H|$ to the aggregate c_p of $\tilde{\mathcal{G}}^B_{c_p,B}(X)$. Turning over to binary or multiclass problems described through one or multiple decision attributes, the relevant relational expressions are outlined by Theorem 4.2.

THEOREM 4.2: *Let be a decision table* $T_{\langle A,D\rangle}, B \subseteq A, E \subseteq D$ *and* $\beta \in [0, 0.5)$*. The* β*-positive region* $\mathcal{L}^\beta_{B,E}(T)$*,* β*-boundary region* $\mathcal{B}^\beta_{B,E}(T)$ *and* β*-upper region* $\mathcal{U}^\beta_{B,E}(T)$ *can be computed by*

$$\pi_{\{c_t\}\cup B}(\sigma_\phi(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^{B\cup E}_{c_p,B}(T))) \tag{4.11}$$

with the condition ϕ *defined by*

$$\phi : \begin{cases} 1 - \frac{c_p}{c_t} \leq \beta, & \text{for } POS_B^\beta(E) \\ 1 - \frac{c_p}{c_t} < 1 - \beta & \text{for } UPP_B^\beta(E) \\ \beta < 1 - \frac{c_p}{c_t} < 1 - \beta, & \text{for } BND_B^\beta(E) \ . \end{cases} \tag{4.12}$$

To obtain the β-regions, (4.11) compares U/B with a more generic version of X/B, i.e. $U/(B \cup E)$, such that those $K \in U/B$ can match with $H \in U/(B \cup E)$ according to B and (4.12). Similar to the previous theorem, $|K|$ corresponds to the aggregate c_t and $|H|$ to the aggregate c_p respectively.

4.4.3 Deducing Traditional Rough Sets

In Section 3.6, it was already clarified that the VPRS model is a generalization of classic RST because the outcome of both is equivalent in those cases where no error is permitted, which means $\beta = 0$ consequently. The same methodology can be applied to

our new in-DB VPRS model in order to deduce traditional rough sets straightaway. In this section, we demonstrate that these expressions of our VPRS model can be further broken down to some simpler versions, which bridges the gap between [186] and [187]. Therefore, let us start off with the approximation for single concepts computing the B-lower and B-upper approximation inside DBs via Theorem 4.3 and 4.4 respectively.

THEOREM 4.3: *Given a data table $T_{\langle A \rangle}$ with $B \subseteq A$ and a concept $X_{\langle A \rangle}$, which is a subset of T, the B-lower approximation of X in classical terms can be computed using the following relational expression*

$$\mathcal{L}_B(T, X) := \tilde{\mathcal{G}}^B_{c_t,B}(T) \cap \tilde{\mathcal{G}}^B_{c_t,B}(X) \; . \tag{4.13}$$

PROOF 4.3: *Considering the equivalence between the classic rough set model and VPRSs in case $\beta = 0$, we have to show: $\mathcal{L}^{\beta=0}_B(T, X) = \mathcal{L}_B(T, X)$. Instantly, we obtain the equalities*

$$\begin{aligned}
\mathcal{L}^{\beta=0}_B(T, X) &= \pi^+_{\{c_t\} \cup B}(\sigma_{1 - \frac{c_p}{c_t} \leq 0}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) && (4.14)\\
&= \pi^+_{\{c_t\} \cup B}(\sigma_{c_t = c_p}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) && (4.15)\\
&= \pi^+_{\{c_t\} \cup B}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_t,B}(X)) && (4.16)\\
&= \tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_t,B}(X) && (4.17)\\
&= \tilde{\mathcal{G}}^B_{c_t,B}(T) \cap \tilde{\mathcal{G}}^B_{c_t,B}(X) && (4.18)
\end{aligned}$$

which justifies Theorem 4.3. □

THEOREM 4.4: *Given a data table $T_{\langle A \rangle}$ with $B \subseteq A$ and a concept $X_{\langle A \rangle}$, which is a subset of T, the B-upper approximation of X in classical terms can be computed using the following relational expression*

$$\mathcal{U}_B(T, X) := \tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \pi_B(X) \; . \tag{4.19}$$

PROOF 4.4: *Considering the equivalence between the classic rough set model and VPRSs in case $\beta = 0$, we have to show: $\mathcal{U}^{\beta=0}_B(T, X) = \mathcal{U}_B(T, X)$. Similar to the previous justification, we get*

$$\begin{aligned}
\mathcal{U}^{\beta=0}_B(T, X) &= \pi^+_{\{c_t\} \cup B}(\sigma_{1 - \frac{c_p}{c_t} < 1}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) && (4.20)\\
&= \pi^+_{\{c_t\} \cup B}(\sigma_{\frac{c_p}{c_t} > 0}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) && (4.21)\\
&= \pi^+_{\{c_t\} \cup B}(\sigma_{c_t > 0 \wedge c_p > 0}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X))) && (4.22)\\
&= \pi^+_{\{c_t\} \cup B}(\tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \tilde{\mathcal{G}}^B_{c_p,B}(X)) && (4.23)\\
&= \tilde{\mathcal{G}}^B_{c_t,B}(T) \bowtie \pi_B(X) && (4.24)
\end{aligned}$$

leading to Theorem 4.4. □

From this point of view, we can describe the B-boundary approximation in terms of DB expressions quite naturally by exploiting the set-theoretic difference between B-upper and B-lower approximation. This is paraphrased by Corollary 4.1.

COROLLARY 4.1: *Given a data table $T_{\langle A \rangle}$ with $B \subseteq A$ and a concept $X_{\langle A \rangle}$, which is*

a subset of T, the B-boundary approximation of X in classical terms can be computed using the following relational expression

$$\mathcal{B}_B(T, X) := \mathcal{U}_B(T, X) \setminus \mathcal{L}_B(T, X) \; . \tag{4.25}$$

Having set the approximation for specific concepts, we consider all of them induced by one or multiple decision attributes in the remainder of this section. Starting with the positive region, a very similar argumentation can be utilized as given in the justification of Theorem 4.3, which guides us to Corollary 4.2.

COROLLARY 4.2: *Given a decision table $T_{\langle A,D\rangle}$ with $B \subseteq A$, $E \subseteq D$, the B-positive region with respect to E in classical terms can be computed using the following relational expression*

$$\mathcal{L}_{B,E}(T) := \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \cap \tilde{\mathcal{G}}^{B\cup E}_{c_t,B}(T) \; . \tag{4.26}$$

Considering the boundary region, we are not interested in those tuples of $\tilde{\mathcal{G}}^{B}_{c_t,B}(T)$ and $\tilde{\mathcal{G}}^{B\cup E}_{c_t,B}(T)$) fully matching but in those with varying cardinality. This is elaborated via the next Theorem 4.5.

THEOREM 4.5: *Given a decision table $T_{\langle A,D\rangle}$ with $B \subseteq A$, $E \subseteq D$, the B-boundary region in classical terms with respect to E can be computed using the following relational expression*

$$\mathcal{B}_{B,E}(T) := \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \setminus \tilde{\mathcal{G}}^{B\cup E}_{c_t,B}(T) \; .$$

PROOF 4.5: *Traditionally, the B-boundary region is received by calculating the complement of the B-positive region (see Definition 3.8). On these grounds, we receive*

$$BND_B(E) = U \setminus POS_B(E) \Leftrightarrow BND^{\beta=0}_B(E) = U \setminus POS^{\beta=0}_B(E) \; . \tag{4.27}$$

In relational terms, this means

$$\begin{aligned}
\mathcal{B}^{\beta=0}_{B,E}(T) &= \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \setminus \mathcal{L}^{\beta=0}_{B,E}(T) && (4.28)\\
&= \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \setminus \mathcal{L}_{B,E}(T) && (4.29)\\
&= \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \setminus (\tilde{\mathcal{G}}^{B}_{c_t,B}(T) \cap \tilde{\mathcal{G}}^{B\cup E}_{c_t,B}(T)) && (4.30)\\
&= \tilde{\mathcal{G}}^{B}_{c_t,B}(T) \setminus \tilde{\mathcal{G}}^{B\cup E}_{c_t,B}(T) && (4.31)
\end{aligned}$$

framing Theorem 4.5. □

Additionally, we would like to point out that Corollary 4.2 (see $\mathcal{L}_{B,E}$) and Theorem 4.5 (see $\mathcal{B}_{B,E}$) can also be contemplated from another direction leveraging Theorems 4.3 (see $\mathcal{L}_B$) and 4.4 (see $\mathcal{U}_B$) respectively. The line of argumentation is worked out in [187] whereas the related expressions $\mathcal{L}_B$ and $\mathcal{U}_B$ were originally introduced in [185]. Yet, neither $\mathcal{L}_{B,E}$, $\mathcal{B}_{B,E}$ nor the relation to Theorem 4.1 and 4.2 were known back then.

4.4.4 Complexity and Implementation Details

In the previous sections, we claimed that computing RST and VPRS inside DB systems is very efficient exploiting relational algebra. In this section, we concentrate on the

runtime and space consideration of Theorem 4.1 and 4.2, which also hold mainly for the derived relational expressions of Section 4.4.3. Additionally, we provide details about the implementation using SQL as query language to express those theorems in practice. Let us start by stating the worst-case runtime employing Theorem 4.6.

THEOREM 4.6: *Considering the computational complexity of the proposed in-DB VPRS model based on extended relational algebra, Theorem 4.1 and 4.2 can be computed with an asymptotic upper bound of* $\mathcal{O}(nm)$ *where* n *is the number of tuples and* m *the number of attributes.*

PROOF 4.6: *On the one hand, the grouping* $\mathcal{G}$ *and projection* π *can be implemented physically using "hash aggregations". The basic idea behind this operator relies on a simple table scan that creates a hash table while traversing the input with grouping or projection attributes as hash key. Buckets in that hash table are either created if the key is missing or statistics of it are updated otherwise, i.e. counts per key in our case (see* $\tilde{\mathcal{G}}$*). Once the scan completes, the outcome of this algorithm is essentially reflected by the hash table containing all desired information, i.e. the specified output attributes and optional aggregates. The dominating costs of the hash aggregation is posed by the table scan, thus, requiring at most* nm *time. On the other hand, the comparison* $\bowtie$ *can be realized by "hash joins". This operator relies on a build and probe phase. By analogy to the previous algorithm, the build phase scans the first input creating a hash table, which is employed in the probe phase to find tuples in the second input with identical key. Corresponding matches are returned. Hence, this technique requires at most* $2nm$*. This way, combining* $\mathcal{G}$ *with* $\bowtie$ *results in* $4nm$*. The selection* σ *can be performed by a sequential scan followed by the final projection constituting* $6nm$ *overall yielding* $\mathcal{O}(nm)$*.* □

Given the justification of Theorem 4.6, we can further elaborate on its space consumption. One can verify that the required memory for the computation is proportional to the size of the constructed hash tables, which leads to the following corollary.

COROLLARY 4.3: *Considering the space complexity of the proposed in-DB VPRS model based on extended relational algebra, the consumed space of Theorem 4.1 and 4.2 has an asymptotic upper bound of* $\mathcal{O}(nm)$ *where* n *is the number of tuples and* m *the number of attributes.*

Given that Theorem 4.6 and Corollary 4.3 rely on adequate hash algorithms with a collision resistant hash function and sufficient main memory for the execution, we would like to state that those operators are implemented by most conventional DB engines. These include but are not limited to Microsoft SQL Server, Oracle DB and PostgreSQL[1]. Moreover, it is worth noting that underlying query plans encapsulating these operators also support a high degree of parallelism, if desired, either given by a single DB node with multiple processors or through a DB cluster reflecting a distributed computing environment. Particularly for DB architectures of the latter kind, we argue that both processing power and main memory are not an issue as more resources can be added conveniently (see Section 2.3.4). These properties are in contrast to other rough set implementations such as given in [192, 193]. In these concrete cases and for many other implementations, the realization mainly relies on sorting that obviously poses a significant higher computational complexity than proposed by our model.

[1] `https://postgresql.org`

Turning to the practical realization of our in-DB VPRS model, we employ prototypical SQL statements following the Oracle SQL dialect. Starting with the β-approximation for a single concept, let be a decision table $T_{\langle A,\{d\}\rangle}$ with the considered target extractable through the selection $X = \sigma_{d=1}(T)$. Thus, Theorem 4.1 can be computed using the query in Listing 4.1 with $B \subseteq A$ with $B = \{b_1, ..., b_m\}$ and $\beta \in [0, 0.5)$. The corresponding *WHERE*-clause specifies the β-approximation.

```
SELECT T.* FROM (
    SELECT COUNT(*) AS c_t, b_1,...,b_m FROM T
    GROUP BY b_1,...,b_m
) AS T JOIN (
    SELECT COUNT(*) AS c_p, b_1,...,b_m FROM T
    WHERE d=1
    GROUP BY b_1,...,b_m
) AS X ON T.b_1=X.b_1 AND ... AND T.b_m=X.b_m
```

$$\textit{WHERE} \begin{cases} 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t <= \beta, & \textit{for } \mathcal{L}_B^\beta(T, X) \\ 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t < 1 - \beta, & \textit{for } \mathcal{U}_B^\beta(T, X) \\ \beta < 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t \textit{ AND} & \\ 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t < 1 - \beta, & \textit{for } \mathcal{B}_B^\beta(T, X) \end{cases}$$

Listing 4.1: SQL statement to compute the β-approximation for single concepts

For Theorem 4.2, we consider $T_{\langle A,D\rangle}, B \subseteq A, E \subseteq D$ and $\beta \in [0, 0.5)$ with $B = \{b_1, ..., b_m\}$ and $E = \{e_1, ..., e_p\}$. It can be implemented by the SQL statement of Listing 4.2. Note, the *WHERE*-clause depends on whether to compute $\mathcal{L}_{B,E}^\beta(T), \mathcal{B}_{B,E}^\beta(T)$ or $\mathcal{U}_{B,E}^\beta(T)$.

```
SELECT DISTINCT T.* FROM (
    SELECT COUNT(*) AS c_t, b_1,...,b_m FROM T
    GROUP BY b_1,...,b_m
) AS T JOIN (
    SELECT COUNT(*) AS c_p, b_1,...,b_m FROM T
    GROUP BY b_1,...,b_m, e_1,...,e_p
) AS X ON T.b_1 = X.b_1 AND ... AND T.b_m = X.b_m
```

$$\textit{WHERE} \begin{cases} 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t <= \beta, & \textit{for } \mathcal{L}_{B,E}^\beta(T) \\ 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t < 1 - \beta, & \textit{for } \mathcal{U}_{B,E}^\beta(T) \\ \beta < 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t \textit{ AND} & \\ 1 - \textit{CAST}(c_p \textit{ AS DECIMAL})/c_t < 1 - \beta, & \textit{for } \mathcal{B}_{B,E}^\beta(T) \end{cases}$$

Listing 4.2: SQL statement to compute the β-region

4.5 Computing Core and Reducts

So far, we sucessfully brought RST and its variable precision extension to the domain of DBs through adequate relational operations. In this section, we motivate why our equivalent RST expressions are also promising for FS and contextual issues of previous approaches (Section 4.5.1). Hence, we postulate necessary corollaries to determine core attributes and reducts with corresponding runtime considerations. For illustration purpose, we partly incorporate them into QuickReduct showcasing an opportunity how to solve FS tasks inside DBs (Section 4.5.2).

4.5.1 The Virtue of Imperfection

In Section 3.5, we briefly introduced FS as an important task to reduce dimensionality and its central role in RST as this theory supplies built-in functionality to address this subject. However, finding meaningful feature subsets is not trivial and so the fusion of FS with DB systems is most encouraging due to efficient operations and data structures provided which we already demonstrated by postulating Theorem 4.6 and Corollary 4.3. Other research activities in this direction fully exploiting DB capabilities are rare though. According to Section 4.2, only the works in [173, 174, 175] are notable. Despite their efficiency, they are not relying on the concept approximation and, thus, convey a crucial drawback, i.e. their limited applicability to inconsistent data. This point leads to two key findings: these methods are less favorable for (i) mining tasks in continuous environments and (ii) reduce FS quality in terms of rough sets. For finding (i), there are real-life scenarios where the removal of inconsistencies is an impractical task or simply not an option. Considering dynamic or near real-time systems with the aim to mine data that become available over time (see Section 2.3), inconsistent entities are of great value and should be kept in the mining process. They constitute an exceptional source to resolve evolving conflicts in the data which we utilize in Chapter 9 in particular. Finding (ii) is not less critical. We argue that imperfection in data can give rise to core attributes, which would not be detectable in environments that are cleaned upfront. To understand the rationale behind this argument, let us consider a DS $\langle U, A, D\rangle$ with the ambition in mind to extract core attributes from A with respect to D. Additionally, let us assume that $\langle U, A, D\rangle$ is partially inconsistent, i.e. $BND_A(D) \neq \emptyset$. Hence, we have two types of classes $K \in U/A$ and $K' \in U/A$: $K \subseteq X, K' \cap X \neq \emptyset$ and $K' \nsubseteq X$ for $X \in U/D$. To seek for indispensable attributes $a \in A$, we examine the partition $U/(A \setminus \{a\})$ and may get $K^* \in U/(A \setminus \{a\})$ with $K^* = K \cup K'$ and $|POS_{A\setminus\{a\}}(D)| \neq |POS_A(D)|$. Thus, attribute a is in the core C reviewing Definition 3.10. One can verify, this situation is not covered when $\langle U, A, D\rangle$ is purged beforehand because K' is inexistent in the consistent case. For a practical example, let us examine the inconsistent decision table $T_{\langle A,D\rangle}$ of Figure 4.1. One can easily determine that $C = \{shape, color\}$. If the inconsistent records x_3, x_5 and x_6 are removed from that table, we only obtain $C = \{color\}$ through this simplification. As the core holds most essential information, this acknowledges that proper reducts cannot be identified and we generally miss the chance to identify important features by ignoring the nature of inconsistency in the data. Hence, a special treatment for data ambiguity can lead to poor results and is not required in RST because it features built-in capabilities to handle uncertainty by definition. As such, inconsistent data can remain in place. Our in-DB model preserves these traditional properties as opposed to others and so proper reducts can be obtained, which is described in more depth in the next Section 4.5.2.

4.5.2 Realization and Complexity

We now formulate the notion of core and reducts based on relational semantics. Let us start with the core holding all indispensable attributes (see Definition 3.10). In line with Theorem 4.2 (or Corollary 4.2 alternatively), core attributes can be computed similarly to the traditional perception given by the following Corollary 4.4.

COROLLARY 4.4: *Given a decision table $T_{\langle A,D\rangle}$ with $B \subseteq A$, $E \subseteq D$, an attribute $a \in B$ is a core attribute with respect to the classification of E if*

$$\mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B\setminus\{a\},E}(T)) \neq \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B,E}(T)) \; . \tag{4.32}$$

On the other side, we can transfer the reduct properties (see Definition 3.9) to the relational domain in addition. Again, we simply can employ Theorem 4.2 (or Corollary 4.2 likewise) leading to Corollary 4.5 that states the two necessary conditions.

COROLLARY 4.5: *Given a decision table $T_{\langle A,D\rangle}$ with $B \subseteq A$ and $E \subseteq D$, $R \subseteq B$ is a reduct with respect to the classification of E if*

$$\mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{R,E}(T)) = \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B,E}(T)) \tag{4.33}$$

and $\forall a \in R$:

$$\mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{R\setminus\{a\},E}(T)) \neq \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{R,E}(T)) \; . \tag{4.34}$$

Note, the usage of the final aggregations $\mathcal{G}$ in Corollary 4.4 and 4.5 is optional. They imply the cardinality of all tuples in the designated positive region, which can be implemented using the SQL query in Listing 4.3. This way, the comparison of both resulting data tables is facilitated. Recall, this technique is already exploited by QUICKREDUCT due to the monotone relation of the positive region in Section 3.5.

```
SELECT SUM(c_t) FROM (
  /*
   SQL subquery to obtain the β−approximation or the β−region respectively
  */
)
```

LISTING 4.3: Statement to compute the cardinality of the β-approximation or β-region

Based on that, let us turn towards theoretic runtime considerations when computing the two aforementioned corollaries inside DB engines. Their worst-case behavior is set out by Corollary 4.6.

COROLLARY 4.6: *For a given decision table with n tuples and m attributes, the costs for (4.32) are $2nm$ for the positive region and two additional scans for the aggregations. Hence, we obtain $\mathcal{O}(nm)$. Consequently, the entire core computation is $\mathcal{O}(nm^2)$ inspecting all m attributes justifying the runtime of Corollary 4.4. Corollary 4.5 in turn requires $4nm$ for (4.33) and $4nm^2$ for (4.34) under the strong assumption that a reduct consists of all condition attributes. Therefore checking an attribute set to be a valid reduct takes $\mathcal{O}(nm^2)$.*

For illustration purpose, we demonstrate the applicability of our proposed in-DB model in terms of FS by simply equipping QUICKREDUCT with equivalent relational expressions. As such, the DB engine carries the heavy lifting of the process to find reducts in the search space and so huge data transports can be avoided as opposed to other implementations. Out of simplicity, we name this ported version QUICKREDUCTDB. It is outlined in Algorithm 4.1. Note, the application of core attributes using our model is deferred to Section 7.3.3.

Input: $T_{\langle A,D\rangle}, B \subseteq A, E \subseteq D$
Output: $R \in \mathcal{P}(B)$

BEGIN
  $R \leftarrow \emptyset$
  LOOP
    $S \leftarrow R$
    FOR $a \in B \setminus R$ **LOOP**
      IF $\mathcal{G}_{\mathtt{sum}(c_t)}(\mathcal{L}_{R\cup\{a\},E}(T)) > \mathcal{G}_{\mathtt{sum}(c_t)}(\mathcal{L}_{S,E}(T))$ **THEN**
        $S \leftarrow R \cup \{a\}$
      END IF
    END LOOP
    $R \leftarrow S$
    EXIT WHEN $\mathcal{G}_{\mathtt{sum}(c_t)}(\mathcal{L}_{R,E}(T)) = \mathcal{G}_{\mathtt{sum}(c_t)}(\mathcal{L}_{B,E}(T))$
  END LOOP
END

Algorithm 4.1: Procedure QuickReductDB to extract a single reduct inside DBs

Lastly, we want to emphasize that Corollary 4.4 and 4.5 are defined on the classic rough set model and not on the basis of VPRSs, which refuses the usage of any tolerated error β. The reason behind this introduced constraint is not relying on the capabilities of our proposed model. In fact, our corollaries can easily be rewritten to allow the parameter β towards the definition of "β-core" or "β-reducts" as given in [186]. Yet, a serious issue may arise because the monotonic behavior of the positive region (see (3.21)) does not hold necessarily in such cases. This complicates the utilization of the positive region as heuristic function. Consequently, it no longer can guide the search for reducts reliably when incorporating variable precision. This impacts QuickReduct, QuickReductDB and many other algorithms that rely on the monotone relation unless $\beta = 0$. To elaborate further on this subject, we refer interested readers to related literature such as [151, 194, 195, 196].

4.6 Comparative Study

This section carries out a comparative study between closely related approaches and our introduced in-DB VPRS model. It is divided into three paragraphs. First, we conduct a comparison from a functional angle by outlining the main characteristics of each system. In the second paragraph, we examine selected approaches in practice by concentrating on the positive region or corresponding metrics if the approach in questions is incapable to compute this region directly. In particular, we measure runtimes required for the respective calculation in a defined test environment and discuss obtained results. In the last paragraph, we state ultimately how our model behaves when more computing resources become available, which corresponds to its scaling potentials demonstrated on shared-everything platforms.

To start the comparative study from the functional side, we contrast our model with the two approaches RSDM [171] and RSMDS [173, 174]. Additionally, we incorporate the established RST frameworks ROSETTA [158], RSES [159] and the software package "RoughSets" [197] for reference purposes. Generally, all of them are capable to compute the approximation for single or multiple concepts except for RSMDS. This point was

Approach/ framework	Concept approx.	Variable precision	Core	Reds.	Uncer- tainty	Client proc.	In-DB proc.
ROSETTA	✓	✓	(✓)	✓	✓	✓	–
RSES	✓	–	(✓)	✓	✓	✓	–
RSR	✓	✓	(✓)	✓	✓	✓	–
RSDM	✓	✓	–	✓	✓	✓	(✓)
RSMDS	–	–	✓	✓	–	(✓)	✓
Our Model	✓	✓	✓	✓	✓	(✓)	✓
✓=characteristic included, (✓)=characteristic partially met, –=characteristic excluded							

TABLE 4.1: Comparison of prominent rough set implementations with key features: concept approximation and regions (concept approx.), variable precision, core, reducts (reds.), the handling of uncertainty (uncertainty), client processing (client proc.) and in-DB processing abilities (in-DB proc.)

already mentioned in Section 4.5.1 because it only exploits properties of the indiscernibility relation rather than to calculate the positive region or any other approximation explicitly. This has implications to the usage of some tolerance functionality too because the introduction of such a variability may jeopardize desired properties in addition. As such, RSMDS is neither capable to use variable precision or any other tolerance measure. This is in contrast to most other systems in this comparison. Turning to the computation of core attributes, there is only an indirect support for this feature in the established RST frameworks ROSETTA, RSES and RSR. By the term "indirect", we mean that these systems just obtain core attributes by performing an exhaustive search for all reducts in order to compute their intersection yielding the core ultimately (see (3.20)). In the documentation of RSDM, we have not found an option at all. The only two models capable to retrieve core attributes based on a given decision table are RSMDS and our solution. However, data have to be cleaned upfront to calculate the core when employing RSMDS and these resulting orthogonal attributes are not necessarily corresponding to those when the data table under investigation comprises inconsistency (see Section 4.5.1). With respect to reducts, all systems deliver adequate solutions but again with the limitation for RSMDS only running on purged data sources. Furthermore, all approaches can be executed on a single client in one way or another. Some points are noteworthy in this regard. While ROSETTA, RSES, RSR are primarily designed for such a setup, RSMDS and our VPRS model are exclusive in-DB processors. This means that the latter two can only be ported explicitly to a client-only model if the underlying DB engine is moved to the client consequently. In this context, RSDM is exceptional because mining partly takes place on the client and at the DB side. However, it turns into a pure client processor on extreme data distributions. This is the case in situations when each produced equivalence class carries one or very few records and so the DB engine is degraded to perform conventional data retrieval operations rather than to conduct mining activities. Lastly, we want to emphasize that ROSETTA, RSES and RSDM provide a graphical user interface to interact with the respective software solution. RSMDS and our in-DB VPRS model are not designed with this purpose in mind. Yet, they are meant to be toolkits that can enrich conventional and distributed DB engines with rough set functionality based on SQL as frequently used utility for data science tasks. A condensed view of these discussed functionalities is highlighted in Table 4.1.

Data set	# of rec.	# of cond. atts.	# of dec. atts.	# of eq. class.	# of class.
HIGGS [198]	11000000	28	1	10721302	2
KDD-Cup 99[4]	4898431	42	1	1075016	23
PAMAP2 [199]	3850505	53	1	3850505	19
Poker-Hand[5]	829201	10	1	829201	10
Covertype [200]	581012	54	1	581012	7
NSL-KDD [201]	148517	41	1	147790	2

TABLE 4.2: Summary of the utilized benchmark data showcasing the number of records (rec.), the number of condition attributes (cond. atts.), the number of decision attributes (dec. atts.), the number of equivalence classes induced by the condition features (eq. class.) and the number of classes induced by the decision attributes (class.)

Given these functionalities, we evaluate how several of these approaches perform in practice by computing the positive region on all available condition attributes. The study focuses on previous solutions except for ROSETTA. Yet, one can draw parallels to RSES and expect a similar output. According to Table 4.1, all approaches provide an implementation for the positive region but RSMDS. Hence, the corresponding metrics of that method are examined instead using the two established DB systems Microsoft SQL Server and Oracle DB. Out of brevity, we name them MsSQL and OraDB for the remainder of this section. Due to the unavailability of RSDM's source code, we redeveloped its basic algorithm in two variants, i.e. an in-DB cursor implementation and a version consisting of a client program implemented using Java that is fed with data from MsSQL or OraDB. Obtained measures for RSES and RSR rely on latest software releases considering the time of writing [186]. With respect to further settings in terms of software, hardware and benchmark data, the experiments are based on a dedicated server environment[2] whereas six data sets are selected from the famous UCI ML repository[3] that are outlined in Table 4.2. Note that all methods and systems are tuned according to technical documentations and best knowledge to ensure fair comparisons. Starting off with RSDM, we utilize the best query plans and most optimal fetch size for its client-server implementation. In this configuration, a high number of network i/o waits is observable in order to synchronize data transmission at the DB side with the processing at the client. As such, the client cannot keep up with the amount of arriving data, which degrades performances particularly for the larger benchmark data sets HIGGS and PAMAP2. Computing the positive region of KDD-Cup 99 is better though which we discuss in a moment. Similar weak performances are unveiled for the in-DB version of RSDM. In neither case, it is able to outrun the client-server realization. On average, the combination of Java and MsSQL is five times faster than the corresponding in-DB cursor variant and pairing Java with OraDB is three times more efficient. Comparing RSDM against our in-DB model reveals that our solutions is almost ten times faster on average considering both DB engines. Yet, the computation of RSDM can generally benefit when the data set of interest is coarse-grained and the

[2] Microsoft Windows Server 6.3 (std., build 9600), Microsoft SQL Server 2014 (developer, 12.0.2), Oracle DB 12c (enterprise, 12.1.0.2), JDK 1.8.0.51, latest JDBC, R 3.2.0 (x64), RSES 2.2.2, RSR 1.3.0, 32× 2.6 GHz Intel Xeon E312xx (Sandy Bridge), 48 Gbyte RAM, 500 Gbyte SAS HDD

[3] `https://archive.ics.uci.edu/ml/`

[4] `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

[5] `https://moa.cms.waikato.ac.nz/datasets/`

underlying partition induced by condition attributes is small. Hence, it can compress the data already inside the DB through its elaborated data retrieval query and so fewer data has to be passed over the wire and to be processed at the client consequently. This can be observed on KDD-Cup 99 most notably where the amount of data is reduced from approximately four million to one million records. Under these circumstances, the advantage of our model is smaller such that it is only 2.50 times faster using MsSQL and 4.60 times quicker employing OraDB. Considering the comparison of RSES and RSR with our in-DB model, we are not taking into account the time of loading the data to memory and only measure elapse times to compute the positive region. RSES showcases the weakest performance and it is significantly outrun by orders of magnitude compared to our VPRS implementation whereas RSR demonstrates a better outcome. On the larger data sets, i.e. HIGGS, KDD-Cup 99, PAMAP2 and Poker-Hand, our solution is at least six times faster than RSR. Especially on HIGGS and PAMAP2, our system has a more than 20 times faster response compared to RSR. These advantages are not that obvious on Covertype and NSL-KDD. Yet, our model is 2.60 times quicker for both DB engines on average. This dominance over RSES and RSR is due to limited parallel processing support such that only one CPU core is exploitable at a time. Our method, though, can use up to 32 CPU cores according to the given hardware profile. At this point it is worth noting that the poor runtimes of RSES are also noticed by its authors such that an extension is introduced in [202] known as DIXER, which permits to distribute jobs to different participating RSES nodes. However, it is questionable if the computation can be boosted in our scenario because calculating the positive region can be considered an atomic task using DIXER. Furthermore, let us examine the practical outcome of RSMDS. It can be setup to utilize very similar hash-based query plans compared to our model. Without preprocessing, i.e. involved data sources are presumed clean, the corresponding expressions to compute core and reduct attributes are very efficient. Its main query to determine core attributes rests upon one projection and one aggregation, while its reduct metric takes twice as much operations. In any case, fewer physical operators are required in contrast to our in-DB model. On this basis, RSMDS generally outperforms our approach. On both MsSQL and OraDB, RSMDS is roughly 30% faster on average, which is due to the additional join operator utilized in our expressions. Yet, taking preprocessing into the equation that is only required for RSMDS, our solution performs better. It is superior by roughly 26% on HIGGS, KDD-Cup 99 and PAMAP2. Considering the processing of the other three data sets (i.e. Poker-Hand, Covertype and NSL-KDD), the difference is marginal with a 10% faster performance on our side. Contrasting both solutions this way, however, neglects the fact that preprocessing activities are usually required only once for a FS task in relation to the actual feature relevance computation and so the runtime dominance of our model is small. A summary of all those reported results is highlighted in Figure 4.2. Note, a further assessment regarding other approximations in the context of classic RST is provided in [187] where we also incorporate runtimes of the DB engine PostgreSQL.

Having discussed the best runtimes, we are eager to take a closer look at the scalability of our VPRS model in addition. Thus, the remainder of this section takes care of this affair by concentrating on vertical scaling scenarios with varying CPU utilization. Both DB engines in our experimental setup support us with this functionality introducing parallel processing for the hash-based query plans (see Section 4.4.4). Hence, processing

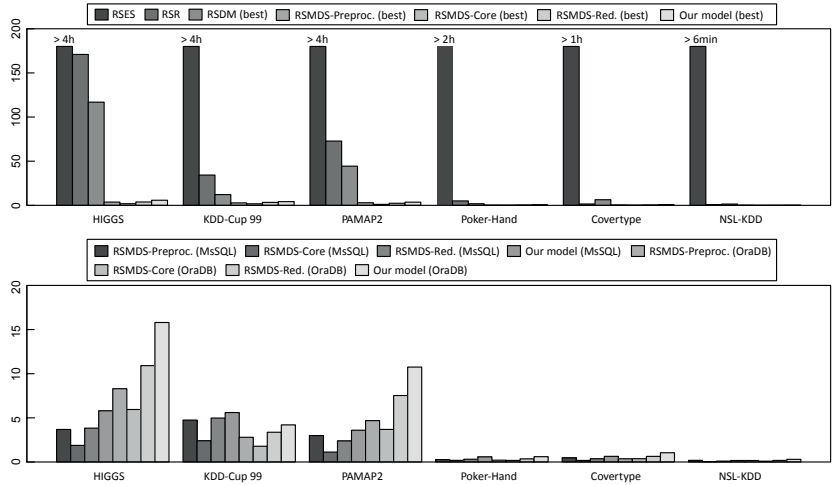

FIGURE 4.2: Runtime comparison of different rough set solutions computing the positive region or corresponding metrics on selected benchmark data sets: best results of all solutions (top) and direct comparison between RSMDS and our in-DB VPRS model (bottom); obtained runtimes are in seconds and reflect the average elapse time of five runs (adapted from [186])

time can be reduced when considering the same problem but employing more CPU power. Therefore, we reuse the benchmark data sets of Table 4.2 and compute the corresponding speedups

$$S_p = \frac{T_1}{T_p}\,, \tag{4.35}$$

where T_1 is the elapse time required to calculate the problem (in our case the positive region on the corresponding data set) using one processor, while T_p is the runtime to solve the problem employing $p \in \mathbb{N}$ processors. Thus, S_p is a measure reflecting the degree how much the actual execution can be improved using parallel processing. The results achieved (constrained under our hardware profile) using MsSQL and OraDB are depicted in Figure 4.3. They indicate that the maximum speedups for the larger data sets on both DB systems are very similar, whereas a better outcome can be manifested for MsSQL on the smaller ones. In this respect, the average speedups using 32 CPU cores for OraDB is 8.40 where the average improvement for MsSQL yields a speedup of 10.60. Yet, one would expect speedups to behave linearly as p is increased, i.e. $S_p = p$, receiving an optimal speedup of 32 considering our environment. In Chapter 10, we get back to this speedup discrepancy and discuss a practical reason for this phenomenon.

4.7 Discussion and Summary

Classic mining frameworks are still a "work horse" for data science and related disciplines today. Even though they comprise a rich functional repertoire, the intrinsic issue

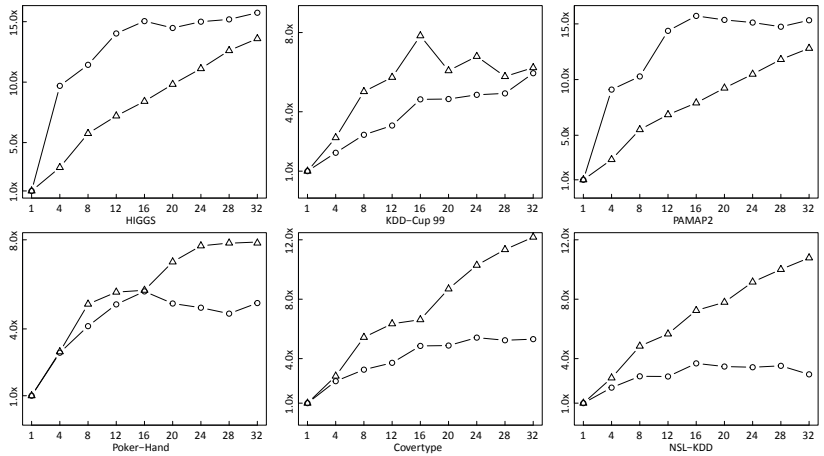

FIGURE 4.3: Speedup of our proposed in-DB VPRS model computing the positive region on different data sets using all condition features; x-axes show the number of CPU cores employed, △-lines and ○-lines indicate the speedup of the systems MsSQL and OraDB respectively; underlying elapse times are based on the average of five runs

of such software systems is their isolation from the problem domain. As such, data must be imported either from the local file system or from remote locations. Loading data this way can be proclaimed time-consuming in the general case, which does not support agile exploratory analysis in the era of big data obviously. A valid argumentation to tackle this concern of classic frameworks is to bring analytics closer to the data and, in fact, the enhancement of relational DB systems with data analysis capabilities in practice is getting under way. This in-DB paradigm is not only promising due to the reduction of vast data loads. Additionally, mining tasks can benefit from very powerful data processing realized by mature DB engines whose algorithms have been constantly improved for more than four decades through the history of these systems. Emerged as a sophisticated instruments for knowledge discovery under uncertainty, RST and extensions match relational DB systems very well conceptually because they are based on set-theoretic operations. In this chapter, we contributed to the trend of orchestrating DB engines for data analysis and proposed a new in-DB VPRS model. By restructuring the concept approximation, new equivalent expressions emerged that can be ported easily to relational algebra and to SQL eventually. Subsequently, we pointed out the context to classic rough sets and derived compliant expressions to determine core and reduct attributes. Lastly, we provided insights to the efficiency of our models from two directions. On the one hand, we formally elaborated on worst-case runtime characteristics. The concept approximation and corresponding regions can be computed in linear time with respect to the number of tuples and attributes in a given table, which is due to the availability of hash-based query plans supplied by most DB engines. This performance is significantly better than those of many RST algorithms relying on sorting operations circulating in the research community. In fact, we are not aware of any other solution computing VPRSs with higher efficiency than our model. The extraction of

the core and determining an attribute subset to be a proper reduct requires at most quadratic time regarding the number of attributes. On the other hand, we contrasted the practical performance of the positive region against other representative RST solutions in a selected testbed. In this setup, our theoretic findings were mainly confirmed. The proposed model is clearly better than conventional implementations and almost ten times faster than RSDM that suffers from heavy network i/o and sorting operations. Under uncertainty, our model even outruns RSMDS by 18% but in cleansed environments RSMDS can take advantage of its elaborated queries, which are 30% faster. Yet, we also stated our model's qualitative advantages over RSMDS as it employs variable precision. In addition, it is capable to handle information deficiencies, which can give rise to most essential attributes for a classification tasks. These cannot be unveiled by RSMDS. As last point of the practical assessment, the scalability of our in-DB model was demonstrated. Using two prominent DB engines, the VPRS implementation obtained a speedup of 8.40 on average for all benchmark data sets. This can be achieved because the mentioned hash-based query plans support a high degree of parallelism, which is a major factor for the efficiency of our model. These scaling capabilities paired with the inherent characteristics of VPRSs and RST to analysis data under uncertainty are of great value. This is particularly true for constantly growing problem domains that are complex in nature. In the chapters to follow, we show that these points are effectively penetrating our endeavor to develop a novel HFIDS. Therefore, our in-DB model is exploited in Chapter 7 to analyze flow features and to extract essential information required for a reliable classification process. Additionally, the model is incorporated in Chapter 9 to propose a new algorithm to extract meaningful decision rules inside DBs by incorporating variable precision.

Part III

Hybrid Flow-based Intrusion Detection

Chapter 5

System Rationale and Design Considerations

In this chapter, we revisit several problems of network security that were already mentioned in the initial chapter of this work. These issues are addressed in greater detail laying the foundation to build a novel hybrid flow-based intrusion detection solution. Additionally, we discuss important design choices from a rather practical perspective. First, we define the threat model of the HFIDS comprised by a number of classic as well as recent attacks. In this respect, we also outline the problem of concept drift and contextualize it along realistic scenarios that may be observed in network systems. Moreover, other operational aspects and practical pitfalls are exposed leading to the selection of flows as primary entity to monitor and analyze network infrastructures. A further operative point is the provision of adequate transparency supporting system operators to understand raised alarms and the behavior of the system. However, existing intrusion detection solutions providing such a transparent view to underlying decision-making carry other essential drawbacks, which finally leads to hybridized approaches by combining misuse-based and anomaly-based detection. Hence, a review of representative concepts and methods is supplied. We show that even these are somewhat flawed with respect to five essential requirements that constitute the cornerstones of our HFIDS. Lastly, the conceptual design of the HFIDS is outlined along with a general overview of involved components that are refined in the chapters to follow.

5.1 Introduction

In Chapter 1, we already gave a brief outline of several major problems that are inherent to the complex domain of network security such that existing solutions aiming at the reliable detection of cyberattacks struggle in one way or another. These problems start by the fact that many NIDSs offer limited scalability because of high computing demands. This way, their applicability to future network systems is questionable given the trend of growing network sizes, traffic volumes and line speeds. Another trend in recent years is that the proportion of encrypted traffic is on the rise for both legitimate and malicious communications. This is a critical factor for those NIDSs that are built upon payload analysis and this point should not be underrated because the amount of these systems deployed in enterprises and institutions is, in fact, predominant. Among those running NIDSs, signature-based engines are very popular due to their deterministic behavior. Once they raise an alarm, operators get the reason for the alert according to the matching signature delivering insights to an attack, which can be leveraged for follow-up actions. Yet, these systems have difficulties to identify intrusion attempts for which no signature is in place and the timely gap between a leaked out vulnerability and the development of adequate signatures covering this security flaw is rather large. This leads to the issue that most NIDSs provide a somewhat limited defense during that time. Hence, there is a demand to reduce the effort to build up signatures and to detect novel attacks at the same time pointing in the direction of a hybrid system combining misuse-based and anomaly-based detection mechanisms. A further concern is that network systems represent nonstationary environments complicating the application of ML for network security that is rarely met by existing IDSs.

In this chapter, we revisit all of these problems in greater detail laying the foundation to build a novel hybrid flow-based intrusion detection solution in this work using ML. First, we highlight classic as well as modern cyberattacks and state their prevalent recency reviewing latest threat reports from recognized enterprises and authorities operating in the cybersecurity sector. Additionally, we outline some prominent incidents that gained media attention in the most recent past. Subsequently, the problem of concept drift is concretized in the context of network intrusion detection along realistic scenarios. This step is very important because, even though cybersecurity is often cited as prime example for nonstationary environments, details are consistently omitted and no case study exists addressing drifts for intrusion detection. Furthermore, we postulate basic assumptions which, combined, constitutes the threat model for the HFIDS to develop (Section 5.2). In a second step, further operational aspects and practical pitfalls are discussed manifesting our motivation to build this system upon IPFIX. We also explain why the employment of flows is a new major challenge for network intrusion detection, which is different to well-known connection-oriented approaches. Another point that is vital from an operative perspective is the provision of adequate transparency given through signatures. However, we also illustrate limits with respect to their intrinsic characteristics and according to recent incidents (Section 5.3). On this basis, we retrospect more promising hybrid intrusion detection solutions and related methods where essentially two main developments can be identified among the large body of literature. In spite of this research effort, several shortcomings can be exposed that need to be minimized in order to render adequate protection for current and future network sys-

tems (Section 5.4). These gaps of existing hybrid solutions paired with the operational aspects mentioned in the previous sections lead to five distinct challenges that require immediate attention. They not only justify our endeavor to build a new HFIDS but also represent the cornerstones of the final implementation. Therefore, we provide a blueprint of the system's conceptual design with all involved modules that are refined in the chapters to follow (Section 5.5). Lastly, a summary is outlined (Section 5.6).

5.2 Threat Model

In recent years, media reports about cyberattacks and resulting data breaches have come thick and fast such that the provision of adequate security measures is more challenging than ever. These developments are of special importance for the design and deployment of an IDS that should be able to keep up with current threat situations while maintaining proficient detection coverage for past attacks that are still plausible. In this section, we, therefore, examine several current as well as classic attacks that build the benchmark for our HFIDS to safeguard network systems. Moreover, we highlight dynamics in the underlying data distribution of networks that are closely related to this threat landscape or just occur as part of legitimate network reconfigurations. Either way, such a nonstationarity poses major challenges for ML (see Section 2.2.3). Lastly, we formulate further important considerations concluding our threat model.

In order to characterize current and classic threat scenarios, a three-stage model is utilized which we already employed in [48]. It abstracts the operational process of an invader sufficiently for our purpose and consists of the following stages that are not necessarily disjoint:

(i) Reconnaissance

(ii) Infiltration

(iii) Exploitation

In stage (i), an attacker identifies and probes its victims. Dependent on the objectives, a number of options are conceivable. "Social engineering" is one of them where, for example, personal assets such as banking details or user credentials are gathered by luring random individuals to visit scam websites via unsolicited bulk phishing campaigns. Similar techniques can also be applied by more personalized attempts known as "spear phishing" or "whaling" that aim for sensible information of a specific target group. Other actions to collect rather technical information about single computer systems or about an entire network infrastructure are classic IP or port scans, which are often referred to as "probing attacks". These can open valuable details of how the network under observation is organized or which machine hosts open ports. As sophisticated tools, vulnerability scanners can be engaged in addition to immediately locate vulnerable systems and services out-of-the-box. This way, the attack surface area can be unfolded significantly. Stage (ii) is concerned with gaining a foothold inside endpoints or complete networks. A straight way is the utilization of stolen credentials obtained previously as part of an identify fraud or by performing "brute-force attacks" against front-ends of web servers or Secure Shell (SSH) services using well-engineered dictionaries that are

traded as commodity in shady online marketplaces. Another option are zero-day exploits targeting unreported vulnerabilities of computer systems. However, there is already a high chance of trapping potential victims with older exploits nowadays. This is due to the huge amount of security gaps and associated software patches that arrive on a regular basis such that an inconsistent update management quickly leads to an insecure system state even though patches are available for these known threats. This is especially the case for web browsers and corresponding plugins that are extremely vulnerable. These circumstances are leveraged by cybercrews in a systematic fashion using so-called "exploit kits" that are at the junction of stage (i) and (ii). Once an endpoint visits an exploit kit infrastructure, the kit first scans browser configurations for specific plugins (e.g. Adobe Acrobat Reader, Adobe Flash Player or Microsoft Silverlight) and tries to find a known security leak in those plugins or the browser itself. As a result, successful matches with available exploits in its back-end DB often lead to an immediate malware infection at the endpoint. These attacks are very effective as the size of malicious code repositories steadily increases and so matches become more probable. With successful infiltration, cybercriminals take advantage of compromised endpoints in stage (iii). In case of a "botnet" scenario where compromised computer systems turn into bots, the application range is versatile. Established "backdoors" in bots, for instance, can be used to receive new instructions such as harvesting confidential data and sending them to the botnet master or to simply allocate computing resources for other undesirable purposes. This way, data can be sold in underground markets or bots can be rented serving as lucrative monetization vector. In particular, the abuse of computing resources can be very harmful when instantiated as part of a "denial-of-service" (DoS) or a "distributed DoS" (DDoS) attack. These attacks aim at disrupting critical services by flooding a system or a whole network with unsolicited traffic loads and, quite naturally, can damage external systems as well as one's own network landscape. In other situations, the malware in charge may not be part of a botnet but represents "ransomware". This malware type asserts another form of monetization by interacting directly with users of the compromised system. Its main purpose is to encrypt valuable documents of the local file system and to extort money from victims in return for the decryption key. Attacks that can be either associated to stage (ii) or (iii) are "SQL injections" (SQLIs) and "cross-site scripting" (XSS). SQLIs are commonly used to either exfiltrate data or to gain access to a system. In both cases, Uniform Resource Locators (URLs) of a vulnerable web servers are manipulated to inject unintended commands to the back-end SQL engine. These can range from simple table reads to content modifications. XSS in turn attempts to inject client-side scripts to a website intended to be executed by all visiting endpoints. As such, various scenarios are plausible. For instance, session information can be hijacked or a script can redirect victims to malicious infrastructures hosting exploit kits. As both attacks occur in the context of web applications, we refer to them as "web attacks". Lastly, we want to mention "man-in-the-middle" (MitM) attacks. Similar to SQLIs and XSS, they can be related to stage (ii) and (iii) where an offender first has to get in between two communication endpoints to finally exploit its position by eavesdropping messages or by manipulating content. One common way to get in this position is achieved by applying "spoofing" techniques.

Having briefly discussed this broad attack spectrum along the operational process of malicious actors, we want to emphasize that, despite their antiquated appearance, most

of these attacks play a central role even in today's threat landscape. With respect to a threat report of 2018 given in [203], it is very common that cybercrime starts off with an unsolicited email and the number of phishing attempts are increasing with more sophisticated content impersonating trusted sources. Once successful, this opens the gate for further offense, which can be depicted by a data breach disclosed in [204] affecting Wipro Ltd.[1], a highly trusted IT outsourcing supplier for enterprises worldwide, in 2019. According to the discloser, all started with phishing attacks trapping a significant amount of computer devices. Attackers then ultimately took advantage of their position inside the Wipro infrastructure to also target several customer networks. Turning over to botnets, they have a long history in the malware ecosystem and their recency can be portrayed by a large-scale incident that occurred at the end of 2016 using the "Mirai" botnet. By performing a coordinated DDoS attack against a Domain Name Service (DNS) provider, Mirai paralyzed Internet services in North America for almost an entire working day [205]. Even though the Mirai creators were prosecuted successfully, botnets have been still very prevalent with respect to another threat report of 2018. Inspired by Mirai techniques, other malware developers are anxious to refine their botnet versions continuously and running them as a rental service. A supplemental insight of this report are advances in the "command-and-control" (C2) communication between individual bots and the botnet masters where encrypted network traffic is becoming the norm rather than the exception [206]. According to a further report for 2019, C2 traffic as well as ransomware were among the top two malware action varieties involved in incidences [207]. Despite this trend, even classic attacks such as SQLIs and XSS are still among the top most critical web attacks (e.g. [208, 209]) and also MitM and brute-force activities should not be underestimated [207]. Particularly, the latter action to ruthlessly break into a system is also highlighted by another report where a growing number of brute-force attacks was experienced in 2019 against the honeypot infrastructure of an renowned anti-malware manufacturer [210].

This threat landscape is further fueled with potential dynamics complicating the reliable detection of attacks as reported in literature. Early works in this direction argue that the behavior of users may change over time raising the demand for adequate adaptability at the detection engine (e.g. [103, 211, 212, 213]). In [211, 212], such changes are declared as concept drift, which can occur on various time scales and also involve the utilization of different software components as outlined in [103]. In this regard, [213] suggest that ML-based detection methods should learn continuously rather than from a static training set to comply with this nonstationarity. Even though we can infer that user behavior certainly is affecting underlying traffic footprints, existing research (e.g. [213, 214, 215]) is sparsely referencing these or other infrastructural changes of a network system as potential drift. While it is mentioned in [213] as a side note, the work in [214] specifically state that adjustments to computing environments are a concern. In that sense, [215] indicate that the traffic within a network is likely to change due to its evolving nature leading to a concept drift problem. Both [214] and [215] further emphasize that in light of adversarial scenarios, adaptability of an IDS is a prerequisite, which corresponds to the view of other authors as well (e.g. [77, 216, 217]). According to [77], intruders tailor their attack vectors to evade detection, which finally might lead to a modification of existing attack signatures as sketched by [216]. [215, 217] report

[1] `https://wipro.com`

about evasion techniques in the specific case of botnet traffic. New instructions received via C2 messages from the botnet master or new binaries replacing older bot versions can change communication patterns and their appearance in the network consequently [215]. Obfuscations are very plausible in addition. For instance, a simple random packet injection to the C2 traffic can undermine security mechanisms [217]. Note, further research discussing evasion tactics with a broader focus on network-based detection can be found in [218, 219, 220].

Given these nonstationary phenomena in the context of intrusion detection, one may ask how these changes evolve over time with respect to their drift characteristics but, in fact, very few details are studied among related literature. Therefore, let us briefly concretize some intuitive situations that might lead to either a gradual, incremental or sudden drift in network systems. Considering benign network activities first, any substitution or update of deployed software components in a computer system that run a distinct network footprint can alter parts of the underlying data distribution as argued above. In modern enterprises and institutions, these changes are usually scheduled tasks coordinated over the network such that all relevant endpoints are updated progressively in a specific time frame. Thus, a transition phase can be inferred where both established and new software installations coexist, which ultimately results in a gradual drift scenario that remains until the complete rollout process succeeds. Note that a similar argumentation holds for the integration of new network segments and the retirement of old ones. Several evasion techniques of an existing attack in turn can be seen slightly more incremental. This can be justified as some of the attack's attributes can be changed by an invader over time but manipulating its inherent traits at once is a difficult undertaking, if not impossible, resulting in a rather stepwise shift. For instance assuming flooding attacks, aforementioned adjustments to the amount of transmitted bytes or packets are actionable evasion tactics. However, data volumes to send cannot be stretched vastly as the intended offense efficiency might degrade. In that sense, factors like the large amount of unusual connection attempts seen in a short period of time are difficult to hide in addition. Other invariant properties are the direct abortion of an established TCP connection triggered by the initiating endpoint as part of a conventional SYN flood or the rejection of exceptionally high connection attempts due to a port or vulnerability scan. Furthermore, abrupt environmental changes require also attention either triggered by an attacker or by legitimate activities. Envisioning the adversary's perspective, their opportunistic attitude paired with a broad attack arsenal may lead to any conceivable sudden drift exposure in principle including zero-day exploits or other unseen exploitation attempts. Yet, such circumstances are less probable in the case of notable legitimate changes. This can be elucidated by the fact that both operators and users aim for controlled network adjustments that appear to be more gradual in nature rather than to jeopardize QoS aspects through radical reorganizations or optimization attempts changing the network configuration abruptly.

Besides these drift considerations requiring a quite liberal learning attitude, some assumptions have to be made. In particular, we constrain invaders to modify portions of normal traffic, which, otherwise, would permit to raise synthetic false alarms to disguise another ongoing attack or to simply overwhelm system operators, i.e. "overstimulation". This excludes MitM or similar intrusions where an attacker attempts to gain control to

legitimate communications for potential data theft or manipulation purposes. In this respect, we also limit ourselves to attacks against network infrastructures only, which means that we neglect adversarial activities against the HFIDS and resilience aspects (e.g. [221, 222]). Moreover, we want to emphasize that our focus is on designing an IDS that is both network-based and centralized in its most basic configuration. This has some implications. As detection is primarily based on network data, attacks can only be uncovered if they provide an ample footprint on network level needless to say that "large-scale coordinated attacks" (e.g. [48, 223, 224]) are neglected deliberately except for DDoS attacks.

5.3 Operational Aspects

In Section 2.1.2, we already mentioned NSM from the early 1990s as one of the first NIDSs that employed packet header information on different abstraction levels to uncover intrusions. From that time onwards, packet-based analysis has been refined constantly to safeguard network assets by incorporating payload data such as given by the systems Snort [50], Zeek[2] [225, 226], PAYL [52] or POSEIDON [227] to name a few. Since then, "deep packet inspection" (DPI) has become state-of-the-art for network security because it can increase detection capabilities tremendously given the access to fine-grained content data. Consequently, it is also exploited in latest commercial solutions from Cisco Systems Inc., FireEye Inc.[3] or Trend Micro Inc.[4]. Despite the success story of payload-based approaches, this type of inspection is resource intensive and a growing number of critical voices has emerged that report about extreme efforts to maintain such systems at moderate line speeds. In [1], operational experience is shared about Snort and Zeek documenting a generally high consumption of hardware resources. This way, arriving traffic bursts can easily overwhelm NIDSs resulting in packet drops and potential attack misses accordingly. These and similar results for open-source NIDSs are in line with assessments of other authors (e.g. [228, 229]) and a survey of CERT/CC[5] [230] revealed a high degree of customer dissatisfaction even with proprietary IDSs in terms of scalability among other issues. In this regard, [6, 231] state that payload-based inspection requires considerably more complex hardware in order to cope with line rates of ten and more Gbit per second (Gbit/s) and [2, 3] even question the scaling potentials of DPI beyond one Gbit/s. Another concern to the practicality of DPI is the increasing amount of end-to-end encryption that is notable nowadays. According to a case study of Sandvine Inc.[6], encrypted global Internet traffic was estimated to reach 70% by the end of 2016 [232] and similar numbers were predicted by Gartner Inc. for 2019 in the context of enterprise web traffic [233]. Hence, the application of payload analysis for intrusion detection purposes is becoming intractable and, therefore, less attractive in the long term unless the implementation of decryption backdoors[7] in underlying proto-

[2] Zeek is the new name of a well-known IDS formerly known as "Bro".
[3] `https://fireeye.com`
[4] `https://trendmicro.com`
[5] `https://sei.cmu.edu`
[6] `https://sandvine.com`
[7] Note, there is an ongoing debate whether law enforcement agencies should be granted access to end-to-end encryption such as given in [234, 235].

cols is pushed ahead. The issue with such a setting, however, is a high risk that the established backdoors can also be exploited as they open up an ideal entry point for attackers as well. Based on these findings, new solutions have to be investigated absorbing these symptoms in the network security sector and a potential remedy points to the utilization of NetFlow/IPFIX (see Section 2.1.1). In fact, choosing flows for intrusion detection purposes has several key benefits:

(i) **Less susceptible to encryption:** Flows are mainly compiled using header information of respective packets. Hence, they are basically resilient to encryption mechanisms of network communications.

(ii) **Fewer load and space-saving:** Flows represent an aggregated view to packet information such that considerable less traffic volumes have to be processed and potentially stored by an flow-based IDS. Concrete savings in comparison to packet data are provided later in this section.

(iii) **Standardized metering and exporting:** The underlying processes of IPFIX are standardized and quite straightforward. Therefore, the deployment of monitoring in network environments is facilitated with respect to operational aspects.

All of these three points address the problems that are at the heart of conventional NIDS and comprehensive DPI in particular. While (i) deals with end-to-end encryption and respects privacy at the same time, (ii) and (iii) are focused on scalability and other sensible points that pop up in practice. On the one hand, network operators usually follow a conservative mindset refusing complex and radical changes to their infrastructures in order to implement monitoring or other additional features. Yet, the fact that over 70% of commercial and research networks are already equipped with flow exporting functionality given through adequate packet forwarding devices (see [236]) facilitate a seamless integration and certainly increase acceptance among those responsible. On the other hand, these flow-enabled switches and routers take care of the heavy lifting aggregating packets to flows that often leads to increasing complexity, packet loss and costs when considering traditional in-line packet capturing methods. As such, it is not uncommon that traffic volume of network lines with ten Gbit/s and more can be handled via one Gbit/s links after the metering and exporting process [6]. Most of these mentioned points are also highlighted by other authors making intrusion detection based on flows the preferable choice to protect high-speed networks (e.g. [237, 238]). From an financial perspective, flow-based intrusion detection also generate a significant lower expenditure compared to conventional solutions (e.g. [6, 231]).

Unlike DPI, flows convey a coarser resolution on network traffic with no or very limited access to content data compared to packet sequences. Hence, it is rather questionable if flow-based detection can reach a similar precision compared to payload-based approaches [237]. In this work, several aspects in this direction are addressed with the desire to build a dedicated flow-based IDS covering a broad range of recent and classic attacks. While, these cyberthreats were already substantiated in the previous Section 5.2, let us discuss essential design choices in terms of utilized tools, standards and corresponding configurations that influence practical experiments of this thesis. Starting with the metering and exporting of flows, various commercial and open-source software tools

Granularity	Avg (1/s)	Max (1/s)	Total (10^6)	Saving (%)
Packets	44023.91	173786	160.17	-
IPFIX uniflows	1466.05	9794	5.12	96.80
IPFIX biflows	777.07	4928	2.75	98.28

TABLE 5.1: Breakdown of a one hour network trace collected at a ten Gbit/s enterprise backbone at different levels of granularity

have emerged such as nProbe [239], Tranalyzer [240], Vermont [241] or YAF[8] [242] aside from hardware-assisted realizations inside packet forwarding devices. We concentrate on software-based implementations with the objective of being able to analyze data as close as possible to NetFlow version 9 and IPFIX. Therefore, we evaluated these and other exporters upfront with respect to their functional scope and their performance characteristics in practice. Results revealed different particularities and strengths. Several of them were very appealing with regards to supplied features and extensibility but their implementations were often limited to a certain degree. For instance, Tranalyzer provided a huge set of flow attributes for monitoring but it was neither directly compliant with NetFlow nor IPFIX. Vermont in turn could not be applied to high-speed networks. The only two exporters worth to consider were nProbe and YAF. While both showed solid performances in practice, we finally selected YAF as it is the reference implementation of IPFIX (e.g [242, 243]) fitting our purpose using default timeout settings, i.e. 300 seconds (idle) and 1800 seconds (active), without sampling. Additionally, we employed the standard five-tuple as flow key and a flow cache dimension equivalent to the main memory capacity of the machine YAF runs on. This way, we move away from NetFlow but emphasize that most practical outcome of this work can also be reproduced using NetFlow version 9 since it is very close to IPFIX from an evolutionary standpoint (see Section 2.1.1). Another crucial point in that sense is whether to utilize uniflows or biflows as basic entity for our analyses. To elaborate on this subject, we assessed the impact of uniflows and biflows in terms of data reduction under realistic circumstances. Therefore, we set up an experiment with access to a ten Gbit/s enterprise backbone from which network traffic was captured for one hour. Comparing the number of packets with exported IPFIX records in that time period disclosed at least a 31.28-fold reduction using uniflows which, on the other end, could even be reduced by almost half taking into account biflows. These results depicted in Table 5.1 not only demonstrate the great space-saving potentials of flows advertised earlier but clearly showcase the advantage of biflows over uniflows in addition. Another amenity of biflows is that all available information regarding forward and backward direction is centered on one flow facilitating data analysis. This way, we can simply identify cases in which, for instance, requests of an originator are actually answered or remain unanswered by the responder (see [40]), which is not directly possible considering uniflows. As such, we select biflows and call concrete instances with a designated feature set "flow vectors" (FVs) to refer to our basic entities utilized to separate attack from benign traffic footprints. Despite their bidirectional characteristics, TCP-related FVs should not be confused with TCP connections though. In practice, state machines are utilized to track connections. As soon as new packets arrive to the corresponding machine, TCP sequence number and involved flags are consolidated to transition from one state to the

[8] `https://tools.netsa.cert.org/yaf/` (version 2.8.4)

next, which often neglects time and hardware constraints. This point distinguishes flows from connections in the general case because the flow metering and exporting take into account timeouts and hardware limits striving for smooth operations. By this means, a flow can be equivalent to a connection under normal circumstances but we can never be sure about that as timeouts or cache flushes provoke early exports such that multiple flows are potentially rendered for a sole TCP connection. Apart from these and other possible artifacts, a further essential distinction is the information carried by FVs. It is organized and encapsulated in new, rather unexplored, flow features that are mainly unavailable in existing connection-oriented studies from the early 2000s and onwards (e.g. [22, 24, 244, 245, 246]). From this perspective, IEs delivered by YAF need to be assessed consequently towards their attack detection capabilities. The list of all considered flow attributes in this thesis is given through Table A.1. Additionally, we would like to note at this point that YAF supplies basic DPI functionality[9]. This is very beneficial to extract application-level information which we exploit to a certain degree as well. Yet, unlike earlier argumentations, we are not interested in analyzing the entire content but eager to inspect only a small amount of initial bytes of a flow's payload in order not to cause any performance issue.

Besides these flow-related considerations, another operational aspect is of fundamental importance. In Section 2.2.2, we already stated that it is essential for several ML-based applications to reach acceptable classification results and interpretability at the same time. Of course, one could argue that such a transparency is dispensable when the model in charge showcases solid predictive capabilities. However, this statement does not hold in real-world environments where the underlying data distribution is subject to change and the model performance is about to degrade sooner or later. As such there are huge incentives to focus on transparency at the very end. Network intrusion detection is one of these problem domains where concept drift plays a central role (see Section 5.2). From that perspective, transparency is also an important design consideration for this work but even if we neglect concept drift for a moment and put ourselves in the situation of responsible personnel, another crucial point towards interpretability is a raised alert by itself. It signals a situation of high severity and requires follow-up activities initiated by system operators in order to assess which vulnerability, if any, was exploited or to which extent an attack impacts the health of the network system. Hence, it becomes clear that ML-based models including anomaly detectors associating nothing more but a predicted class label to an incoming example (in our case a FV), i.e. the exclusive treatment of a classification problem, are not sufficient. According to [15], such "pseudo IDSs" are a true concern because in the past researchers oftentimes misused the intrusion detection domain as playground to evaluate proposed ML method rather than deploying proper solutions that respect operational aspects. A very similar valuation is given in [14] where the authors refer to a "semantic gap" caused by these unexplainable black box predictions. In that sense, raised alerts are providing simply too little context to guide operators towards the actual meaning of an alarm and to assess what are necessary next steps. Thus, it is not surprising that the majority of proposed ML-based IDS solutions are not achieving a maturity level above laboratory usage. In fact, predominant systems found in practice are signature-based engines (e.g. [14, 16, 17]). From our point of view, one of the main reasons

[9] `https://tools.netsa.cert.org/yaf/yafdpi.html`

for the success of these IDSs is certainly the simplicity of signatures that contribute to the overall transparency in analogy to Section 2.2.2. Once an alarm is raised by these systems, an operator directly can investigate on the incident by reviewing the signature details narrowing down potential causes and impacts. Besides this concrete advantage, other authors additionally argue for signatures because they facilitate interoperability with other systems as they can be easily shared. Ideally, they are independent from each other such that the introduction of new signatures or minor modification of existing ones can be done without interfering existing parts and, ultimately, they have a relative low false alarm rate (e.g. [17, 49, 225]). While these benefits are very useful, the last mentioned point leads to a paramount drawback of signature engines in practice. In order to reach a low level of false alarms, signatures must be very strict in terms of their coverage. These conservative bounds in turn make it less likely to hit a new attack or to unveil slight variations of an existing one, i.e. a generally high miss rate for attacks with no rule in place. As such, the utilization of evasion techniques or zero-day attacks can simply bypass signature-based security mechanisms rendering these systems ineffective until an appropriate rule becomes available by the IDS manufacturer or by the research community. To get an impression about the severity of this problem, let us consider the evolution of the "WannaCry" attack as an example that broadly became known in 2017. Apart from classic ransomware behavior, it also exhibited a worm-like spreading mechanism that brought an unprecedented new level of quality to malware deployments at that time. By exploiting vulnerability CVE-2017-0144[10] among others, WannaCry infected computers running specific versions of the operating system Microsoft Windows around the world up until recently although a patch[11] for these systems was released by March 2017. In that time period, WannaCry signatures[12] also became available for Snort. Beside its destructive power and the mentioned patch and signature releases, it is interesting to know when CVE-2017-0144 actually was unveiled by attackers. A reliable date is obviously difficult to obtain even though there are unconfirmed speculations that the National Security Agency[13] was aware of this vulnerability for more than five years right before WannaCry gained media attention [247]. Anyway, it is known with certainty when the security leak in charge was issued. The official registration dates back to September 2016, which manifests a period of at least six months where endpoints and network systems virtually were vulnerable to that threat and so the attack remained undetectable by Snort and similar signature-based systems in that time slot. Another example is the more recent CVE-2019-0708[14] where an intruder can take advantage of a Remote Desktop Protocol (RDP) security gap permitting remote code executions. This vulnerability, known as "BlueKeep", was registered at the end of November 2018 while Snort rules[15] were not available before May 2019. These and other examples demonstrate the susceptibility of signature-based engines to unknown attacks not least because of the laborious creation process of rules that typically involves manual intervention. This deficiency, however, is the advantage of anomaly detection engines. To be successful with respect to the detection of novel intrusions, these systems oftentimes follow a "closed-

[10] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144/`
[11] `https://support.microsoft.com/en-us/help/4013389/title/`
[12] `https://snort.org/advisories/talos-rules-2017-04-25/`
[13] `https://nsa.gov`
[14] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0708/`
[15] `https://snort.org/advisories/talos-rules-2019-05-20/`

world assumption" [14, 99]. In our context, this means that any observation that is not even close to what we initially defined to be normal must be malicious as a logical consequence. While appearing to be sound at first sight, complex problem domains such as given through network intrusion detection let these systems frequently fail in practice by producing too many false alarms (e.g. [14, 78]). Hence, a straightforward notion is the combination of both anomaly-based and misuse-based detection. This idea is not new after all and has been investigated intensively in literature. Consequently, a review of related concepts is provided in the next Section 5.4.

5.4 Related Hybridizations

Research attempting to combine misuse and anomaly detection dates back to the early and mid 1990s with the proposition of the hybrid systems IDES and its successor NIDES [21, 248]. Both systems leverage a rule-based expert system and a statistical anomaly engine running in parallel. As they are mainly intended for host-based analysis, promising results evolved into [249] as distributed IDS with the primary aim to protect large-scale networks. Apart from the IDES family, several approaches follow that design such as [22, 23, 250, 251, 252]. Misuse and anomaly detection mechanisms are discussed in [22]. Yet, available algorithms only operate on specific data. While misuse methods are applied on network and host data, normal profiles just can model user behavior. This way, both detector types need to run in parallel in order to get the most out of this tool set. The work in [23] uses a self-organizing map as anomaly component and extracted rules from the DT C4.5 [59] serving misuse detection. A "decision support system" takes care of the final decision since potential disputes of both detection modules need to be handled accordingly. A similar approach is employed in [250] to detect DDoS attacks exclusively. The system uses MMs and Snort. Additionally, both detectors are intended to share information to update the corresponding model but details are omitted. In this respect, updates to the signature engine highly depend on new hand-crafted rule releases from the developing community of Snort. A rather static version of Snort is also applied in [251] in combination with multiple ML algorithms. The work in [252] either can utilize Snort or any other rule engine and engages C4.5 as anomaly detection module. Remarkable to this approach is the continuous training of C4.5, which relies on data previously classified serving adaptivity aspects. Additionally, it operates on flow-like data in combination with raw packets.

Another methodology is the sequential formation of both detection mechanisms. The first notable prototype in that direction is introduced in [24, 253], which examines incoming traffic against Snort. If no signature fires at this system component, data are passed over to the custom anomaly detector. Detected anomalies at this stage are finally translated into signatures via "automated signature generation" (ASG) and integrated into Snort for upcoming classifications. The authors in [25, 254] also use this methodology to relax the high computational costs of the anomaly detector. Its detection engines rely on random forests and the overall architecture is clearly split by an online and offline part. In the online phase, incoming traffic is classified by the misuse detectors and, in case no rule applies, turned over to an offline buffer, which is the source for an outlier classifier. This buffer is necessary for the overall architecture

because the authors are aware their anomaly component is too inefficient for real-time traffic processing. Finally, rules are extracted from identified anomalies and attached to the online misuse component. In [244], a very similar approach is conducted in comparison to [24, 25, 253, 254]. All of them employ connection-oriented features to detect intrusions. In particular, it should be noted that due to the selected features in [24, 253], detection capabilities tend to overfit. This is partly confirmed by the authors of [26] as an extension of [255]. They are inspired by the design and the drawbacks of [253] at the same time, i.e. a promising architecture but a too generic approach causing many attack misses and false alarms for real-world scenarios. Therefore, the authors suggest to incorporate payload analysis to build signatures being convinced that this kind of ASG mediates a much higher predictive performance. In addition, an anomaly model based on Hypertext Transfer Protocol (HTTP) specifics is proposed to detect protocol violations. This is not a direct restriction to web attacks because modules for other protocols can be implemented too. Nevertheless, it can be considered a laborious task to develop such anomaly models for each protocol in order to obtain a general-purpose IDS. Other hybrid systems operating in a cascading fashion and capable to induce signatures automatically are [256, 257]. However, the problem of [256] is its limited focus tracking code-injection attacks on host level only. On the other end, [257] operates on network data but its rule generation relies mainly on suspicious traffic, which means that the data source for the induction is constrained and cannot take into account sufficient benign data to create high quality rules. Anyhow, this system stand out as rules are created for both normal and malicious traffic footprints. Three more proposed hybrid methods are noteworthy: [245, 246, 258]. The technique in [258] is exclusively designed to protect web servers given log files and engages anomaly detection first followed by misuse approach to get a manageable amount of false alarms. In [245], a tree structure produced by C4.5 is leveraged as misuse approach that can unveil malicious activities in a transparent manner, while special SVMs are trained at leaves with benign decision implementing the anomaly detection part. The last work introduced in [246] sticks out because it actually represents a mixture of parallel and cascading processing. Initially, a clustering runs as anomaly detector. If an anomaly is identified at this stage, a second anomaly detector is incorporated to either confirm or to reject the hypothesis of the first and an instance-based method is applied serving as misuse component in case the clustering predicts a normal activity. All of the three mentioned approaches neglect ASG.

Other approaches concentrate on ASG alone without a necessary system architecture underneath. Their scope of application is diverse. Several proposed methods run on host level to find vulnerable programs and to extract signatures for malware footprints (e.g. [259, 260, 261]). Further ASG focus on worm outbreaks in networks (e.g. [19, 262, 263]) or exploit honeypots as source to create appropriate signatures (e.g. [18, 20, 264]). While this branch of research has generated a solid methodical foundation, most methods operating on network level utilize intensive payload analysis. From a practical point of view this is problematic given the increasing employment of encrypted traffic nowadays (see Section 5.3). Hence, a promising direction is to develop ASG solutions based on established flow standards that are less sensible to encryption but, according to [265] and our experience, misuse detection on flow level is rather underdeveloped. We are aware of only three notable works: [266, 267, 268]. [266] generally assess whether packet-based

alerts of Snort can be reproduced on flow level modeling their correlation as learning task. In turn, a signature engine classifying IPFIX records is proposed in [267]. A benchmark comparison with Snort shows that the introduced system is capable to operate at much higher network speeds than Snort and detects all involved attacks. Two drawbacks can still be identified, i.e. only HTTP related rules are considered by the authors and all of them are hand-crafted. Therefore, [268] tries to automate the rule extraction. Relying on a labeled data set of flows and computed features, ML-based methods are exploited to extract rules with the idea to translate them to Snort signatures. Results on a recent benchmark data set demonstrate solid detection rates for several attacks. This indicates that signatures built on top of flows can be beneficial but research must be extended to other data sources in order to confirm obtained findings.

Recapping this overview of related concepts, we observed that a large body of research deals with the hybridization of intrusion detection in various aspects. Most proposed systems attempt to leverage the benefits of misuse and anomaly detection. Early solutions try to combine both complementary approaches by using them in parallel. While many solutions of that kind confirm fair results, minor attention is attributed in terms of transparency and ASG during operations. This trend changed when research employed misuse and anomaly detection sequentially with the desire to turn expose anomalous footprints into signatures that are certainly more interpretable. Either way, solutions of both research directions carry essential deficiencies challenging their practical usage. They are rarely assessed towards scalability aspects, which leaves open whether these systems meet contemporary throughput capabilities to safeguard current and future network landscapes. Straight scalability may only be attested for [249] and [256] according to their distributed architecture although the system proposed in [249] has no support for ASG and [256] is neither suited for network data. Another serious concern is the limited applicability in nonstationary environments for most systems, which is a prohibitive assumption in our target domain. The only exception is [26] due to the selected detection mechanism for HTTP-based traffic that is invariant to drifts. To some extent, the works in [25, 254] can also be considered adaptive to drifts as long as completely labeled training data sets become available over time. While this is rather unrealistic due to high labeling costs (e.g. [13, 77, 78]), the approach in [252] is worth to mention in this context. It is the only hybrid flow-based approach in our review that can adapt its underlying model implicitly to changing circumstances. Though, it is doubtful if this system runs reliably on sudden changes in the network because its anomaly detector is solely based on C4.5 in a supervised setting. Moreover, utilized features in [252] tend to overfit learned models and insufficient support for transparency and ASG is apparent. Apart from these downsides, methods with an exclusive focus on ASG have been proposed as well. Yet, those solutions handling network data mainly concentrate on specific situations such as worm outbreaks and heavily rely on payload analysis, which comprises other practical pitfalls.

5.5 Conceptual Architecture

The current threat landscape paired with the stated operational aspects and existing solutions disclose that further progress is imperative for network security research despite

the huge effort made over the years. In particular, advances towards the hybridization of different detection mechanisms are very promising. Yet, proposed systems either operate primarily on host level (see [21, 43, 256]), focus on specific attacks (see [26, 250, 258]), have a restricted scalability (see [25, 246, 254, 257]), are not applicable under concept drift (see [24, 244, 245, 251, 253]), offer limited transparency and ASG functionality (see [23, 249, 252]), cannot be applied to encrypted traffic scenarios (see [19, 20, 26, 264]) or are simply incapable to detect novel attacks based on network data (see [22, 267, 268]). In this work, we contribute to these research needs and attempt to close open gaps of existing solutions by proposing a new HFIDS. Our system is new with respect to the combination of properties and challenges that shape the key pillars of the HFIDS as follows:

- **Flow monitoring:** As flow-based IDS, the primary data source of the HFIDS are flows based on the standardized IPFIX protocol. In particular, we select biflows due to a number of advantages over uniflows. In association with all other pillars, the fundamental challenge is to find meaningful flow features covering our threat model heading towards general-purpose intrusion detection.
- **Transparency:** This aspect is vital from an operations point of view. However, it is clear that the construction of signatures for novel attacks is difficult. Hence, a mechanism must be established that rapidly transforms opaque security alerts into patterns describing regularities in the data such that reoccurring alarms are available in an interpretable format.
- **Scalability:** In order to comply with speed demands of current and future networks, the HFIDS must rely on an architecture that scales in the sense that higher loads to the system can be compensated by appending more hardware resources. Hence, algorithms to develop and utilized frameworks must be designed appropriately.
- **Adaptivity:** Drifting circumstances in terms of legitimate and attack behavior are intrinsic challenges for network security. These cause uncertainty from a learning perspective and degrade classification performances sooner or later. Consequently, these situations must be anticipated by the HFIDS requiring appropriate adaption capability. This point is complicated considerably when available training data with underlying GT are sparse.
- **Reliability:** The combination of signature-based and anomaly-based approaches is an aspirational target because the benefits of both can be combined. This also includes the ability to produce a low false alarm rate, which is of particular importance in our problem domain suffering under "class imbalance" where attacks are far less observed in comparison to legitimate traffic. This challenge is directly connected with the utilization of flows as main data source.

By seeking for an architecture with the potentials to combine all of these five pillars, two basic directions exist to fuse misuse and anomaly components. In the first, authors are suggesting to run both components in parallel. Approaches of that kind have an essential disadvantage because they are not providing sufficient transparency. This is due to the fact that transparency is only met in situations where both detectors successfully predict an attack. In all other cases where an alert is raised by either detector, a dispute needs to

be resolved. This in turn adds undesirable complexity to the decision-making process. More encouraging is the other direction where a cascading architecture is harnessed, which is employed in this work as well. In particular, we adopt the basic design of [26, 244, 253, 254]. In what follows, we point out the main conception towards a novel HFIDS.

Picturing a high-level view on the basic architecture of this centralized NIDS, let us start with the data ingestion, which is the first step in the process chain and called "data preprocessing and correlation" (DPC). It expects flows to arrive from selected IPFIX exporters installed in the infrastructure to monitor. In addition to this primary source of data, we also permit to fuse flows with supplemental information widening the context. Particularly, we consider specific host events collected at distinguished computer systems in the network requiring a correlation of both the independent stream of events and flows. By choice, DPC is also able to extract meta data by relating similar flows to each other in order to capture the situation surrounding them. Thus, the result of this initial step are assembled FVs carrying flow features and optional non-native flow information. These prepared entities are passed over to the "pattern matching component" (PMC) that compares each arriving FV against existing patterns located in the "pattern DB" (PDB). In this specific context, we decide patterns not only to describe malicious footprints but also to characterize benign regularities in the data, which separates them from signatures in the classical sense (see Section 2.1.2). Both forms of patterns can be generally abstracted by decision rules. As such, the match of an attack rule to an incoming FV leads to an alarm reported to the "alert management" (AM) along with the fired pattern. If a benign rule can be associated, the FV is simply persisted at the "traffic DB" (TDB), i.e. a repository containing recently consumed FVs. However, further actions are imperative if no pattern matched at all. In this case, FVs have to undergo a final inspection at the "anomaly detection component" (ADC), which either raises alarms for suspicious FVs or just stores processed benign FV in TDB. Lastly, the content of TDB has to be analyzed keeping PDB up to date. This is performed through the "pattern building component" (PBC), which complements the HFIDS. At this stage, data residing in TDB are either utilized to induce new patterns if existing knowledge in PDB was not proficient enough to explain FVs just processed or TDB is leveraged to generalize familiar patterns in PDB. An overview of all building blocks of this proposed HFIDS is illustrated in Figure 5.1. Even though this described two-stage detection approach is very intuitive, incorporating the five pillars is not straightforward. Flow support is certainly enabled by DPC. Yet, the coverage of a broad attack spectrum as well as providing reliability in terms of few false alarms not only requires meaningful features but also sophisticated learning algorithms. This includes a sound interplay between ADC and PBC as well because misjudgments at ADC have an direct impact to the quality of interpretable patterns. Likewise the learning strategies of both components must be adaptive to address concept drift. If such dynamics cannot be anticipated completely, the system should at least supply mechanisms that facilitate the interaction with operational staff at AM to resolve adaptivity problems conveniently. As the large-scale processing and handling of flows can be considered to be in big data terrain [6], the entire process chain of the HFIDS must be distributed well among different computing nodes. At the same time, communication across system boundaries has to be minimized to finally fulfill scalability demands.

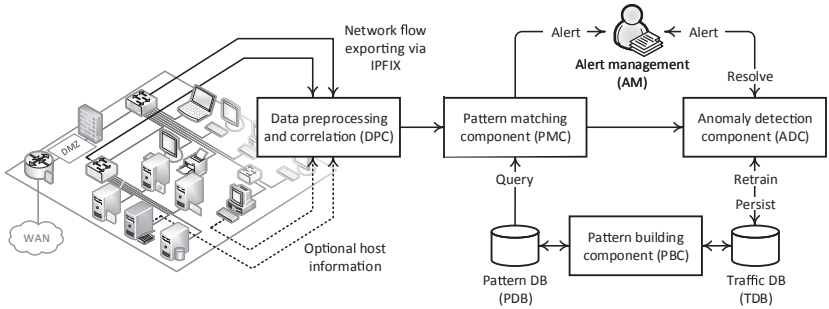

FIGURE 5.1: Proposed architecture: prototypical network landscape to monitor (left) and basic building blocks of the HFIDS (right)

5.6 Summary

In this chapter, we refined several problems that are affecting the field of network security along with basic assumptions that lay the foundation to propose a new centralized NIDS in this work. We first started by outlining the current threat landscape intimidating owners of network sites and involved users. In this light, we discussed the challenge of nonstationarity. We argued why it is an inherent problem for network intrusion detection and contextualized the general definition of concept drift with our target domain. Moreover, we drew attention to DPI as frequently employed method in recent intrusion detection products and emphasized that its applicability is increasingly becoming more cumbersome with growing network line speeds and the rise of end-to-end encryption. While the former point might be compensated with extremely expensive hardware equipment, the latter point is not that easy to overcome. From a technical perspective, traffic monitoring based on flow technology such as NetFlow/IPFIX delivers a profound and affordable remedy for both points but it is still unclear to which extent information comprised by flows can cover current and future threat situations. Beside this open issue, we debated system transparency as key factor for network security operations. Hence, signature-based systems are deployed predominantly rather than anomaly detectors. This point is serious because these systems offer limited protection for evasion tactics or novel attacks which we concretized by most recent cases. To obtain a transparent intrusion detection solution capable to detect a wide spectrum of known and unknown attacks, a logical choice directs to a combination of both system types. Therefore, a literature review was conducted on several representative hybrid intrusion detectors, which exhibited two main directions that are flawed in one way or another. On these grounds, we postulated five challenging aspects relevant in practice that address the problems of those mentioned systems and proposed a concept of a new HFIDS with these fundamental aspects at heart. This blueprint is sharpened gradually in the chapters to follow towards a concrete implementation.

Chapter 6

The NDSec-1 Network Traces

In light of the recent evolution of cybercrime postulated as part of our threat model, reliable security measures must evolve in the shape of sophisticated NIDSs to protect network infrastructures. Designing and building such a detector is highly data-driven in order to uncover malicious traffic footprints. This in turn requires adequate reference data supporting different phases in both the development and deployment process. However, finding publicly available benchmark data sets reflecting realistic network situations with a proper GT is an undisputed challenge, which also affects our ambition building up the proposed HFIDS. Many existing data sets are either outdated or focus on very specific subjects such as botnet, flooding or brute-force traffic rather than providing a broad repertoire of different attack vectors threatening today's networks. Bridging this research gap, this chapter contributes a new attack composition comprising a multitude of classic as well as state-of-the-art attacks. The data set embrace rich and untreated packet captures including payload, collected log events and a detailed GT. Initial qualitative assessments on these proposed traces reveal that they are a solid supplement to existing benchmark data sets rendering a well-founded base for mining applications in the field of network security research.

This chapter is based on:

F. Beer, T. Hofer, D. Karimi, U. Bühler: "A new attack composition for network security". In: P. Müller, B. Neumair, H. Reiser, G. Dreo Rodosek (eds.) *10. DFN-Forum - Kommunikationstechnologien*, Lecture Notes in Informatics, vol. P-271, pp. 11–20, GI, 2017.

6.1 Introduction

Given our threat model, the advances of today's cyberattacks against network infrastructures are versatile and alarming. Consequently, there is a huge demand for trustworthy remedies. Research in this direction frequently utilize ML techniques to build sophisticated intrusion detectors (see [269, 270]). The downside of these approaches is the demand for quality data sets in order to train models and to secure their validity in practice. Hence, such benchmark data ideally have to fulfill several requirements to meet general-purpose characteristics: (i) they should reflect realistic and unprocessed traffic from the underlying network system to safeguard. (ii) Supplied data should be embraced by an open and fine-grained format, which offers additional opportunities such as mining meaningful features to increase detection capabilities or to benchmark competing solutions that rely on different inputs including packet or flow data. (iii) The data set should cover a multitude of classic as well as state-of-the-art attacks that are plausible in the environment of interest. (iv) Moreover, it should provide a detailed GT annotating traffic with class labels indicating both malicious as well as benign data instances. (v) Ultimately, log information residing at host level should be contained. These supplemental data either have the potentials to enrich the prediction process with valuable information that are out-of-sight for conventional sensors in a network or simply can support researchers with appropriate meta data generating more context. Given these intuitive requirements, yet, it is a challenging task to find such a data set that is publicly available, which is a true concern in the network security community with respect to repeatability and comparability of research works [28]. This depressing situation often forces researchers to rely their work on reference data assembled more than two decades ago. Actions in this direction are obviously questionable as many attacks of that time are obsolete and taken assumptions are no longer valid as a matter of fact [27] such that further progress in network security research is strongly impeded. Thus, there is an urgent need for quality data sets that is constantly reported in literature (e.g. [27, 28, 29, 34, 271]).

Motivated by these findings that are also affecting our ambition to build a new HFIDS, we introduce a new data set in this chapter. It is termed "NDSec-1" and intended to be shared among the network security community. Hence, it has to meet the requirements postulated in the previous paragraph in order to be beneficial for a broad range of purposes. While most of these demands can be obtained with diligence, accomplishing aspect (i) is not straightforward. The reason for this relies on the fact that the rendering of realistic legitimate network traffic is highly dependent on the underlying infrastructure including influencing factors like software configurations and human interaction (see Section 5.2). Consequently, benign network data cannot be produced in the general case and, therefore, should be collected at the target domain instead. This is particularly true for typical anomaly-based intrusion detection setups applying the concept of "normality" (see Section 2.1.2), which in turn is reliant on the network system to protect. With this perception in mind, we argue that malicious footages are of high interest because many existing network captures, in fact, focus on isolated attack types such as botnet, brute-force or flooding. This is very specific for a general assessment of a NIDS. Therefore, we assemble a data set primarily concentrating on attack data rather than on legitimate traffic using state-of-the-art penetration testing suites, malware instances collected "in

the wild", recently reported exploits and classic tools. As this arsenal is very basic equipment for cybercriminals, we carefully incorporate it into well-defined scenarios reflecting realistic attack situations, which are applicable to most conventional network infrastructures. Furthermore, our data set embraces untreated packet traces including payload and captured log events documented by a rich GT. Thus, it can be reused to "salt" legitimate traffic based on common strategies such as the "overlay methodology" [272] supporting several important steps to build a NIDSs.

The remainder of this chapter is structured as follows: first, we review commonly used benchmark data sets employed in the field of network security (Section 6.2) followed by the introduction of our new data set (Section 6.3). On this basis, a qualitative evaluation is performed by contrasting the functional scope of our data sets with those proposed by other authors. We also provide practical insights applying our captures to a well-known NIDS (Section 6.4). Ultimately, we close this chapter by discussing and summarizing obtained achievements (Section 6.5).

6.2 Common Benchmark Data Sets

Even though critical voices emerged over time complaining about the absence of adequate benchmark data sets, in fact, several attempts have been made over the last decades. The most prominent and recognized are the DARPA 98/99 traces [30, 31, 32] and derived data set versions such as KDD-Cup 99[1], NSL-KDD [201] and GureKDDcup [273]. Despite their age, these are frequently employed in research works although widely criticized due to design flaws and their inability to meet contemporary requirements with respect to traffic and state-of-the-art attack variants (e.g. [201, 274, 275]). Early succesors like Kyoto2006+[2] never reached that degree of acceptance in the research community. Other more recent traces deliver quality data, but pursue specific objectives. SSH-DS [11] and L-Flows [33] mainly focus on brute-force attacks supplying highly aggregated flow data, while CTU-13 [34] and Booters-15 [35] either concentrate on botnet traffic or on "DDoS-as-a-Service". Regarding multi-purpose intrusion detection, the only suitable data set comprising a broader attack range is ISCX-2012 [276] considering the time composing our original research in [36]. The problem with ISCX-2012, though, is that it only contains few malicious traffic especially when taking into account flow-based intrusion detection such that its application for both training and testing a detector can become unfavorable in practice. Concurrently to [36] and afterwards, several other data sets evolved, namely, NGIDS-DS [277], CIDDS-1 [278], CIDDS-2 [279] and CICIDS-17 [280] that are of particular interest for multi-purpose attack detection in addition. The assembling of existing data sets can be a further source of valuable data by combining independent network traces in order to obtain desired characteristics. Two of these collections are ISOT [215] and ISCX-Botnet [281] that mainly concentrate on botnet data. On the one hand, ISOT incorporates real captures from LBNL/ICSI[3] and the TrafficLab at Ericsson Reasearch [282] with malware traces from the Honeynet

[1] `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
[2] `https://www.takakura.com/Kyoto_data/`
[3] `https://icir.org/enterprise-tracing/`

project[4]. On the other hand, ISCX-Botnet partially merges traffic from ISOT, ISCX-2012 and CTU-13 to compile a representative data set. Lastly, we would like to mention "capture the flag" competitions such as CDX-2009[5], DEFCON-CTF[6] or iCTF[7] where a high load of malicious activities can be tapped. According to [283], they are gaining importance to build realistic benchmark data sets although past contests revealed a number of shortcomings such as inadequate network topologies, sensor placements or regulations that inhibit their usage. Note, a recent survey of other data sets for network intrusion detection is given in [284].

Unfortunately, there has never been a generally accepted data set for network security after the "golden days" of the DARPA 98/99 traces although several efforts are noticeable in the community to provide captures for multi-purpose intrusion detection. They started in parallel to our research with very similar ambitions, which stresses the urgency for quality benchmark data sets in addition. It is exciting to see these developments such that progress in network security research can regain momentum. As an extension to our effort in [36], we illuminate the basic characteristics of these brand-new data sets as part of our comparative study in Section 6.4. Yet, they cannot be incorporated beyond this chapter due to the availability of those captures released just recently.

6.3 Infrastructure and Attack Scenarios

Based on the discussed absence of appropriate traces fulfilling intuitive and fundamental demands, we propose a new benchmark data set in this section. As opposed to most other attempts, it contains few background traffic and, thus, pursues a rather unorthodox solution, i.e. an attack repository, from which one can pick and choose to enrich other captures. Furthermore, we attach importance to other practical aspects that jointly interact with the requirements postulated in Section 6.1. These can be summarized along the following principles:

- Support of various attack types and variants
- Attacks wrapped around realistic scenarios
- Simple infrastructure to incorporate other traces
- Rich GT based on biflow semantics
- Offer raw packet captures including log events

To organize the description of our NDSec-1 traces, we first highlight the underlying network infrastructure of that data set (Section 6.3.1) and detail involved attack scenarios (Section 6.3.2). Finally, the attack and packet distribution is outlined (Section 6.3.3).

[4] https://honeynet.org
[5] https://westpoint.edu/centers-and-research/cyber-research-center/data-sets/
[6] https://defcon.org/html/links/dc-ctf.html
[7] https://ictf.cs.ucsb.edu

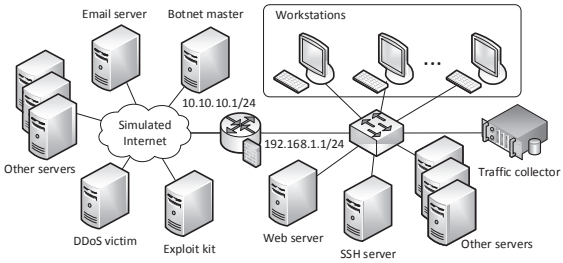

FIGURE 6.1: Simplified network landscape to build up NDSec-1: the simulated Internet (left) and the protected private network (right) (adapted from [36])

6.3.1 Network Infrastructure

To build up an appropriate infrastructure for NDSec-1, we follow a conventional network setup that is placed on a virtual testbed[8] located inside our campus network. Running as hypervisor, this testbed mimics two subnets, i.e. a private network representing the enterprise or institution to safeguard and the simulated Internet, constituting a simple network topology. In order to obtain an overall realistic traffic behavior for that topology, an OpenWRT[9] router is engaged controlling the communication between both subnets in a very specific manner. On the one hand, this router acts as Network Address Translation gateway with firewall capabilities for the private network. Only port 80 is opened to redirect ingress traffic to the internally hosted web server of that subnet. On the other hand, the router is configured in such a way that real Internet requests are forwarded to the campus network. The hardware inventory of the simulated Internet consists of prepared "virtual machines" (VMs) such as an email system, an exploit kit or a botnet master serving as controlled infrastructure for most attack scenarios. The private side in turn relies on a heterogeneous set of workstations and servers installed with recent versions of Microsoft Windows and Linux. Traffic capturing is conduced by a typical tcpdump[10] sensor inside the private network. Hence, only ingress and egress traffic of that subnet is observable by that sensor reflecting primary audit data. As supplemental source, we also collect log event information (i.e. syslog [285] and Microsoft Windows event messages) locally at each host which we extract after each attack scenario completes. An overview of this very basic but common network topology is outlined in Figure 6.1.

6.3.2 Attack Scenarios

In this section, we outline three distinct attack scenarios that are contained within NDSec-1. Moreover, we repeated all earlier intrusion attempts and performed several other attacks without specific context in a fourth experiment.

[8] VMWare ESXi 6.5.0 (build 4564106) on a Fujitsu Primergy RX200 S7: 2× Intel Xeon CPU E5-2630 (2.30 GHz), 96 Gbyte RAM, 4× 1 Tbyte SAS HDD configured as RAID-5

[9] `https://openwrt.org`

[10] `https://www.tcpdump.org`

Bring your own device: The pragmatic "bring your own device" (BYOD) mindset is increasingly applied in enterprises and institutions. It authorizes employees and partners to utilize personal hardware inside the organization's network infrastructure with access to privileged resources. Despite all of the advantages provided by this liberal policy, new security risks arises, which allow an attacker to act from the inside. We exploit such a plausible situations in this scenario by placing a machine to the private network constituting a compromised BYOD. Due to an installed backdoor supplied by Metasploit[11] using a binary Linux trojan, full access to this device is granted. Within this preliminary setup, several reconnaissance activities are performed against the infrastructure in order to study the unknown network. Ultimately, two potential victims are identified, i.e. an internal SSH server and a workstation observed to frequently connect to both an email and a web server. To gain a foothold inside the former machine, a dictionary brute-force attack is carried out disclosing valid login credentials successfully. In addition, we target the latter endpoint to steal sensible information in two possible ways. On the one hand, we maneuver ourselves in between workstation and email server silently by combining Address Resolution Protocol (ARP) and DNS spoofing techniques. Based on that, we leverage this advantageous position pretending to be the legitimate server forcing the endpoint to key in sensitive account data which we later misuse to take over the related email account. All of these MitM actions are not attracting much attention through elaborated redirections leading the victim to believe it connects to the legitimate email server which, in fact, is the case right after we trap the data. On the other hand, a similar technique is pursued to breach a connection but this time between victim and web server. As the communication is end-to-end encrypted, we engage ARP spoofing and SSLsplit[12] to establish a Secure Socket Layer (SSL) proxy. Using this approach is highly sophisticated because we can perform a MitM attack regardless of the HTTP Secure (HTTPS) communication by mimicking the server's SSL certificate on-the-fly among others to finally steal valuable information from that session. To complete the data theft in this scenario, several hijacked assets are uploaded from the BYOD to an external File Transfer Protocol (FTP) server in the simulated Internet.

Watering hole: It is not uncommon for enterprises or institutions to self-host services from within their infrastructure making them available from the inside and the outside. In this scenario, an external intruder tries to compromise an internally hosted web server with the intention to infiltrate and later exploit a related group of workstations, i.e. a "watering hole attack". Therefore, we perform a brute-force attack against the front-end of the web server in the first place using Medusa[13] followed by an SQLI to retrieve logins and password hashes from the back-end DB of the hosted web application. Reusing this gathered information, XSS is employed to inject client-side scripts to private pages of users found in the DB permitting to target a small group of users that are likely to visit these pages of the website. The intention of these placed scripts is a malicious redirect to an external exploit kit scheme, i.e. Crimepack 3.1.3, that we prepared and situated in the simulated Internet upfront. In our case, the kit is configured to exploit an unreported Microsoft Internet Explorer vulnerability to infect visiting endpoints with the ransomware "ToxiCola". Note, each instance of this specific malware contains cus-

[11] `https://metasploit.com`
[12] `https://roe.ch/SSLsplit`
[13] `http://foofus.net/goons/jmk/medusa/medusa.html`

tomized binaries generated by the malware author right before it is deployed, which certainly complicates detection. The results of this scenario are successful infections on two endpoints inside the protected network. On this basis, ToxiCola encrypts several important local documents residing at each endpoint in order to request money in exchange for decryption details. Additionally to these blackmail attempts, ToxiCola is also reporting back to a known server in the Internet.

Botnet: We already highlighted that the rental of botnets operated by cybercrews is a lucrative business in the underground economy (see Section 5.2). Hence, these illicit infrastructures are increasingly gaining popularity. This trend is crucial for enterprises and institutions because essentially any host of a legitimate network may turn into a bot and, thus, has the potentials to participate in an criminal act unintentionally once infected. "Citadel" version 1.3.5.1 as revised variant of the well-known "Zeus" botnet is employed in this scenario. Based on a normal operating network, we infect three legitimate workstations with Citadel binaries. To reach this point, the infection is conducted through two classic attack vectors. On the one hand, conventional email spam lures two users to open malicious attachments targeting vulnerability CVE-2015-2509[14] (Microsoft Windows Media Center) and CVE-2015-5122[15] (Adobe Flash Player). On the other hand, the third infection is caused by a rogue download that is triggered by visiting a compromised website in the simulated Internet. As such, all three bots are connected to their prepared botnet master via HTTP. Among characteristic C2 footages between master and bots, we commission all bots to download new instructions. These contain malicious payload to perform a synchronous SYN flooding attempt to a single destination outside the network. Aside from this successful DDoS attack, two of the bots are also issued to steal local configuration files, which are later transfered to an external FTP server hosted in the simulated Internet.

Attacks without specific context: In this experiment, all attacks from the previous three scenarios are repeated without specific context. Additionally, we perform a number of other attacks. For instance, we use the tool Yersinia[16] to run DHCP starvation attacks, which exhaust the number of available IP addresses from a known DHCP server utilizing spoofed Media Access Control (MAC) addresses. HTTP floods are carried out using the Apache HTTP server benchmarking tool[17]. Additionally, we seek for vulnerabilities with Nikto[18], i.e. a toolkit particularly designed to scan web applications, and target an FTP service with the well-known THC Hydra tool[19] to extract user credentials by brute-forcing the hosting server. Tsunami[20] is employed to perform DNS amplification attacks resulting in a DNS flooding attempt. Finally, we make use of the classic hping3[21] to send a high amount of User Datagram Protocol (UDP) packets to specific target hosts in the network.

[14] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2509/`
[15] `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5122/`
[16] `https://github.com/tomac/yersinia/`
[17] `http://httpd.apache.org/docs/2.4/programs/ab.html`
[18] `https://cirt.net/Nikto2/`
[19] `http://sectools.org/tool/hydra/`
[20] `https://samiux.blogspot.com/2014/07/tsunami-dns-amplification-attack-tool.html`
[21] `http://hping.org/hping3.html`

Attacks	# of packets	bytes per packet
Botnet (Citadel)	5198	707.27
HTTP brute-force	26093	495.12
FTP brute-force	1530	63.01
SSH brute-force	20873	179.04
HTTP flooding	167238	115.68
SYN flooding	890895	100.54
UDP flooding	2275614	137.46
Malware/exploits	8802	866.76
Probing	21707	329.15
Spoofing	1199	60.01
SSL proxy	11602	776.83
SQLI/XSS	334	278.54

TABLE 6.1: NDSec-1 attack and packet distribution (adapted from [36])

6.3.3 Recap of the Captures

As a result of the processed scenarios inside the simplified testbed, numerous attack types and variants are covered through NDSec-1. These can be encapsulated by 12 broader categories. A summary of them along with the captured packets and byte distribution is illustrated in Table 6.1. Note, most of the malicious actions were performed manually. Particularly, the execution of the defined scenarios was carefully crafted to evade detection as much as possible. Thus, we believe the resulting data set reflects realistic attack footages. In order to provide an adequate GT for NDSec-1, we first converted each involved network capture to biflows using YAF and employed this coarse format for the manual labeling process. This way, each flow is either assigned according to its underlying attack category or to the legitimate class otherwise. In order to share the outcome of all conducted experiments among the network security community including all raw packet traces, local log files and rich GT, we published all relevant information at our website:

`https://hs-fulda.de/NDSec/NDSec-1/` .

Again, we would like to highlight that NDSec-1 is designed to represent pure attack sequences with a low focus on generating legitimate background traffic because such benign characteristics are better suited to be collected from within the network system to safeguard. To obtain a quality data set with both attack and benign data in place, we, therefore, recommend to salt legitimate network traces with our captures, which is a technique already suggested in [271]. Since the basic data format of NDSec-1 is very fine-grained, this data set may support the evaluation of existing or new network intrusion detection solutions that are either based on packet, connection or flow data.

6.4 Qualitative Evaluation

Given these details about NDSec-1, qualitative aspects of this benchmark data set are outlined next. First, we supply a comparative study of involved characteristics between several related traces found in network security literature and NDSec-1 (Section 6.4.1).

Second, we share initial practical results by applying NDSec-1 to a recent version of the signature engine Snort and highlight to which extent attacks comprised in our scenarios can be detected by this state-of-the-art IDS (Section 6.4.2).

6.4.1 Comparative Study

In this section, we consider the ten benchmark data sets Booters-15, CICIDS-17, CIDDS-1, CIDDS-2, CTU-13, ISCX-2012, ISCX-Botnet, L-Flows, NGIDS-DS and SSH-DS. These are compared to NDSec-1. For this purpose, we start off with the format, which is an intrinsic property of all data sets because it determines for which purpose the designated data set is applicable. The majority of captures in this section provide rich packet traces (see "pcap format") permitting the most general applicability as they can be used to analyze packet header, payload, connections or flows (i.e. Booters-15, CICIDS-17, CTU-13, ISCX-2012, ISCX-Botnet, NDSec-1 and NGIDS-DS). Other data sets offering a coarser format are limited by definition. For instance, CIDDS-1, CIDDS-2, L-Flows as well as SSH-DS only delivers highly aggregated flow data. As such, they prevent examinations below this level of granularity. Hence, the latter sources can only be utilized for the emerging field of flow-based intrusion detection (see Section 7.2). In this context, the applicability is also affected by the attack diversity held in the data set, which either permit to assess the performance of an IDS towards a broad attack range or to isolated malicious activities. NDSec-1, ISCX-2012 and CICIDS-17 are among those embracing the largest attack repertoire, while others contain significant fewer amounts of intrusions or aim at specific attacks exclusively. Moreover, the underlying environment is an essential characteristic. Several of the examined data sets were captured in the wild or under equivalent conditions constituting most realistic traffic (i.e. L-Flows, SSH-DS, Booters-15 and CTU-13). However, network traces with these qualities usually include sensible information raising privacy concerns. Therefore, some of these evaluated traces are made available on flow level only, were anonymized or sanitized such that certain information in these data sets is lost or become ineligible (e.g. anonymized IP addresses for CIDDS-1, CIDDS-2 and SSH-DS or cleared payload for CTU-13 on benign data). Note that, we refer to the term "raw data" if the observed captures are not postprocessed. To circumvent privacy issues, an alternative are "synthetic" data sets, which are network traces recorded in a controlled environment (physical or virtual infrastructure). Yet, this does not necessarily mean that they are inappropriate or less qualified to train or validate network intrusion detection solutions. Indeed, this type of data sets is gaining more attention in literature (e.g. [271, 281, 286, 287]) and can produce realistic traffic footprints once the environment is setup properly. Traces comprising this characteristics are CICIDS-17, CIDDS-1, CIDDS-2, ISCX-2012, ISCX-Botnet, NDSec-1 and NGIDS-DS. The last property we want to stress is the provision of an appropriate GT, which is another critical point for existing data sets as outlined in [28]. Network traces carrying both malicious and legitimate traffic generally require a GT to apply supervised ML that may exist on different levels (e.g. per IP address (see ISCX-Botnet) or flow (see CICIDS-17, CIDDS-1, CIDDS-2, CTU-13, ISCX-2012, L-Flow and NDSec-1)). Simple annotations distinguishing between normal and attack instances obviously suffice binary classification problems, but in-depth assessments of captures can only be achieved using rich labels including attack types and additional remarks. The latter is covered only

Data set	Available format			Raw data	Synthetic	Involved attacks								GT		
	pcap	Flow	Log			1	2	3	4	5	6	7	8	IP	Flow	Rich
Booters-15	✓	–	–	(✓)	–	–	–	–	✓	–	–	–	–	–	–	–
CICIDS-17*	✓	–	–	✓	✓	✓	✓	✓	✓	✓	–	✓	–	–	✓	✓
CIDDS-1*	–	✓	✓	(✓)	✓	–	✓	–	✓	✓	–	–	–	–	✓	✓
CIDDS-2*	–	✓	✓	(✓)	✓	–	–	–	–	✓	–	–	–	–	✓	✓
CTU-13	✓	–	–	(✓)	–	✓	–	–	✓	✓	–	–	✓	–	✓	–
ISCX-2012	✓	–	–	✓	✓	(✓)	✓	✓	✓	✓	–	✓	–	–	✓	(✓)
ISCX-Botnet	✓	–	–	(✓)	✓	✓	–	–	✓	✓	–	–	✓	✓	–	–
L-Flows	–	✓	✓	–	–	–	✓	–	–	✓	–	✓	–	–	✓	✓
NDSec-1	✓	(✓)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	✓
NGIDS-DS*	✓	–	✓	✓	✓	–	✓	✓	✓	✓	–	–	✓	–	(✓)	(✓)
SSH-DS	–	✓	✓	–	–	–	✓	–	–	–	–	–	–	–	–	–

*=inexistent at the time of writing [36], ✓=characteristic included, (✓)=characteristic partially included, −=characteristic not included; 1=botnet (C2, fraud, fast flux, etc.), 2=brute-force, 3=other malware/exploit, 4=flooding (DoS and DDoS), 5=probing, 6=spoofing, 7=web attack (SQLI, XSS, etc.), 8=others (spam, SSL proxy, etc.)

TABLE 6.2: Comparison of most related intrusion detection data sets

by the five data sets CICIDS-17, CIDDS-1, CIDDS-2, L-Flows and NDSec-1. In this regard, SSH-DS and Booters-15 are not providing such a GT at all because all involved data refer to malicious traffic. The depiction of all discussed criteria given for those considered data sets is outlined in Table 6.2.

Retrospecting the time of writing [36], several points can be manifested. Most data sets at that time were captured with specific goals. L-Flows and SSH-DS comprise flow data and log event information, which can enrich the detection process by opening further insights to potential attack situations. However, the limiting factor on both captures is their underlying data format restricting a detailed analysis below flow level such as DPI. On the flip side, Booters-15, ISCX-Botnet and CTU-13 comprise rich traces. They concentrate either on malware or DDoS-as-a-Service traffic ignoring other sophisticated attack vectors including brute-force or web attacks. These findings are crucial particularly when working towards a general IDS embracing various attack types. In that time, the only two data sets enclosing a wider range were ISCX-2012 and NDSec-1. Yet, ISCX-2012 neither covers log events nor a detailed GT, which heavily aggravates examinations per attack. Moreover, it does not incorporate frequently used spoofing attempts, MitM attacks or real botnet traffic as opposed to NDSec-1. Beyond that time, several other data set emerged with similar aspiration to ours. In this respect, the two closest approaches to NDSec-1 are CICIDS-17 and NGIDS-DS but neither of these traces reach the attack diversity of NDSec-1. Additionally, CICIDS-17 does not supply log information and the GT of NGIDS-DS is somewhat convoluted.

6.4.2 Practical Insights Using Snort

As opposed to ML techniques that are examined extensively in Chapter 7, this section briefly reports about the qualitative results running NDSec-1 against the well-known system Snort. Therefore, we use version 2.9.9 with default system settings, latest community rules and the emerging threats (ETs) extension[22] and collect all alarms per

[22] `https://rules.emergingthreats.net` (version 8499)

attack scenario that are raised by Snort. For the purpose to diagnose matches and to get conclusive insights, we map these rendered alerts to the corresponding flow-based GT of NDSec-1 utilizing timestamps, IP addresses and ports. Within this simple setup, let us start with the BYOD scenario. Results reveal that most of the probing attempts in this scenario remain undetected particularly for the frequently used port range 0 to 1023. However, Snort is capable to discover some vertical scan activities for specific ranges, i.e. port 5800 to 5820 and port 5900 to 5920. Additionally, some signatures throw alarms for DB-related ports providing meaningful messages. The SSH dictionary attack in this scenario is alerted on a periodical basis including the SSL proxy, while neither the covert channel to our remote machine nor the ARP spoofing attempts are exposed by Snort. With respect to normal traffic, some minor false alarms are noticeable especially on the alert message "PROTOCOL-DNS TMG Firewall Client long host entry exploit attempt" occurring in approximately 7% of all benign cases. Since such a security gateway is not installed in our environment, this message is misleading and dismissed consequently for all involved attack scenarios to follow. Within the watering hole scenario, a high amount of traffic is generated to perform the HTTP brute-force in order to get a foothold into the corresponding web server. Yet, no signature is triggered on this activity. A similar behavior is observable for the XSS placement and traffic produced by the injected ransomware disclosing that related infections remain undetected. Those SQLIs as well as traffic induced by Crimepack in turn are unveiled. Eight different ET signatures apply to the former such that around 40% of the injections are uncovered. Traffic caused by the latter is unmasked entirely by an ET alarm designed for the "Eleonore" exploit kit stating that Crimepack employs similar footprints. Turning to the botnet scenario, exploited vulnerabilities (i.e. Microsoft Windows Media Center and Adobe Flash Player) and C2 traffic are identified with explicit ET signatures. However, involved DDoS and data theft actions remain stealthy. This comes at no surprise for the latter because the exfiltration relies on a legitimate FTP upload but within a rogue context. This is obviously difficult to determine using signature-based engines. Applying the last scenario without specific context, results disclose a very similar outcome for attacks intersecting with the previous scenarios. Nevertheless, flooding attacks based on HTTP and UDP basically remain undetected, while FTP brute-force and vulnerability scans can be identified in roughly 33% and 63% of the cases.

These observations confirm that several attack instances inside NDSec-1 can be safely uncovered utilizing Snort with default settings. On the flip side, some basic attacks are missed or hit only partially. Particularly, the latter attacks comprise a high traffic volume compared to other sure detections (see Table 6.1). Taking this factor into account, the overall classification on flow level discloses a poor outcome in terms of conventional metrics such as hit or error rate (see Section 7.4.1). Being aware that a more sophisticated configuration including enterprise rules would certainly boost Snort's detection abilities, obtained results using NDSec-1 look very encouraging yet. This manifests that the incorporation of penetration testing suites, recent malware instances and classic attack tools within realistic scenarios provide a sound base to support development cycles of a new detector or may uncover weaknesses of already deployed IDSs.

6.5 Discussion and Summary

Network security research is highly data-driven and, thus, depends on high-quality reference data to assess existing or new intrusion detection techniques. Unfortunately, obtaining such a data set is not a trivial undertaking. Actual research works often resort to benchmark data compiled more than 20 years ago, which is doubtful given the latest advances of the cybercrime ecosystem that were not known back then. Even affecting our work, this depressing situation certainly hinders further progress in network security research. In this chapter, we worked towards a practical solution for this problem by providing a novel data set that is shared among the research community. We started by postulating several intuitive requirements with the intention to fulfill general-purpose characteristics. In this regard, it was figured out that it is rather difficult to supply realistic legitimate traffic because such data are largely dependent on the underlying network infrastructure to safeguard especially when considering typical anomaly detection approaches. Therefore, we attached importance to the generation of attack data instead that can be fused with legitimate traffic of one's own network site. These attack data were carefully crafted and wrapped around realistic scenarios incorporating penetration testing suites, recent malware instances and classic attack tools inside a generic topology that basically matches any other network system. In that sense, we targeted the network from the inside via a compromised BYOD that we controlled remotely using a backdoor. Additionally, we performed a multi-stage attack against the legitimate infrastructure by first infiltrating an internally hosted web server from the outside that later served as infection vector to implant ransomware to visiting hosts of that network. Furthermore, we took over several workstations using conventional spam turning them into bots. This way, we could exfiltrate local assets and exploited those systems to participate in a criminal act by shutting down an external victim using a DDoS attack. Lastly, we leveraged the network infrastructure to perform other attacks without specific context. To evaluate these scenarios with respect to a state-of-the-art NIDS, we applied our data set to a recent version of Snort. Results were two-fold. On the one hand, several comprised attacks could be safely unveiled by this system demonstrating a solid outcome. On the other hand, some very basic intrusion attempts remained undetected or were hit only in parts, which manifest the merit of our methodology combining classic and modern attack tools towards a sophisticated and broad repertoire. In particular, it is this specific characteristic alone that distinguished our attempt from related solutions considering the period of time when our data set was actually composed. Having said that, it is still very competitive even for today's standards as examined qualitatively in our comparative study of latest network captures found in literature. Moreover, the data set comprises unprocessed packet captures, a detailed GT as well as log information. Combined, this makes the provided traces a distinct attack repository and a profound supplement for the network security community. In fact, an increasing interest has been notable based on the number of conducted correspondences and inquiries seen since our initial publication. Up until now, more than 20 research institutes from more than 15 different countries have requested our proposed benchmark captures including universities from the United States, China, Germany, Japan, Singapore, Sweden and Poland. These observations not only confirm our effort but further encourage us to employ the traces extensively during our journey to build up our HFIDS sketched in Chapter 5.

Chapter 7

Flow Feature Analysis

With the proliferation of the flow standard IPFIX and NetFlow as its predecessor that are available in many packet forwarding devices nowadays, flow-based intrusion detection became more pronounced. In recent years, results in this direction have been very promising. Despite these efforts, existing methods mainly focus on specific detections such as the uncovering of botnet or brute-force attacks. Only few attempts exist that are concerned with the effectiveness of flow technology for general-purpose intrusion detection. Hence, little is known about valuable flow features and their capability to classify a multitude of attack types reliably. This chapter concentrates on these gaps by performing a flow feature analysis towards the identification of meaningful attributes mainly derivable from NetFlow/IPFIX data. This is conducted using our in-DB rough set model. As a result, several minimal feature subsets emerge that are evaluated in terms of their practical credibility using ML techniques. Additionally, we study the combination of flow features and log events. Results on independent benchmark data sets disclose fair predictive capabilities for the selected feature subsets in relation to the basic learners employed. Performances can be further increased by taking into account statistics derived from traffic movements in addition to pure flows.

This chapter is based on:

F. Beer, U. Bühler: "Feature selection for flow-based intrusion detection using rough set theory". In: G. Fortino, M. Zhou, Z. Lukszo, A. V. Vasilakos, F. Basile, C. E. Palau, A. Liotta, M. P. Fanti, A. Guerrieri, A. Vinci (eds.) *Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC 2017)*, pp. 617–624, IEEE, 2017.

7.1 Introduction

Recalling the argumentation of Section 5.3, a direct comparison of packet data and flows in terms of their attack detection potentials reveals an intrinsic disadvantage for flows. By definition, they consist of highly aggregated information and so fewer indicators are at our disposal to guide a reliable detection in contrast to DPI that is based on a fine-grained data source. With the increase of encrypted traffic and the advances of high-speed networks, the advantage of payload-based analysis, however, vanishes. Hence, the focus shifts back to flows pinning hopes on scalable intrusion detection for current and future infrastructures. They are less susceptible to encryption and space efficient. Furthermore, traffic monitoring can be integrated into existing network landscapes straightaway using the flow standard IPFIX or NetFlow that are supported by a growing number of packet forwarding devices. These very practical features have stimulated research and so flow-based intrusion detection has become a hot topic since the mid 2000s. During that time, several sound solutions evolved such as to detect brute-force or botnet attacks. Undoubtedly, these approaches address parts of the current threat landscape but represent point solutions at the very end. Only few attempts exist to either fuse already existing approaches or to compile a new system towards general-purpose flow-based intrusion detection capable to unveil both state-of-the-art and classic attack types. In this respect, one of the main factors impeding further progress on this subject is not only related to the limited availability of data exclusively as stated earlier in this work (see Chapter 6). In fact, several flow-based benchmark data sets have been introduced most recently. The problem is also related to find a coherent data basis to select valuable flow features because the number of intersecting features on these independent data sets turns out to be very small due to their heterogeneous structures[1]. Hence, it is easy to identify useful features on one benchmark data set but it becomes very difficult to confirm such findings empirically with others. Given these observations, it is not surprising that intensive flow feature analysis has been omitted in the research community for the most part. Indeed, this subject requires further attention because recent works only center on feature analysis for botnets not considering other types of attacks such as spoofing, brute-force or man-in-the-middle. From this perspective, it is not clear whether flow-based intrusion detection using NetFlow/IPFIX technology is, to some degree, an alternative to payload-based detection despite the mentioned benefits in practice. In this chapter, we make an attempt to reduce gaps in recent research by performing a comprehensive feature analysis towards their effectiveness to cover a broad range of different attacks. Therefore, we contemplate IPFIX data as primary data source and use extracted statistics of traffic movements as complementing dimension to assemble a list of ground features. From this list, we seek for meaningful feature combinations by applying a FS algorithms that builds upon our in-DB rough set model. As a result, several feature subsets are distilled that can be considered minimal to separate benign and malicious footprints. To verify the operative effect of these condensed sets, diverse ML algorithms are applied on independent benchmark data too. In this context, the impact of security-related logs residing on host level is studied in addition to pure flows

[1] This point underpins our earlier argumentation to publish raw packet traces instead of aggregated data sets permitting to extract new features also at a later stage after the data have been made publicly available.

and derived statistics. These are considered supplemental information and correlated to existing FVs with the intention to further boost predictive performances. Combined, this course of action can be seen as preliminary study for the final HFIDS architecture because it lays the foundation to build reliable FVs that serve as input for ADC and PBC eventually.

To cover these points, we organize this chapter by giving an overview of representative achievements on flow-based intrusion detection using NetFlow/IPFIX first (Section 7.2). On that basis, we discuss considered traffic traces and preprocess them to finally obtain meaningful flow features using FS (Section 7.3). These feature sets are assessed subsequently with different ML methods towards their applicability on prepared benchmark data (Section 7.4). Ultimately, we give a brief wrap-up of conducted activities and gained insights (Section 7.5).

7.2 Flow-based Analysis Using NetFlow/IPFIX

Flow-based intrusion detection using NetFlow/IPFIX has attracted researchers for years. As a result, methods emerged analyzing network probing, worm traversals, DoS or policy violations [288, 289, 290, 291], which are successfully combined in CAMNEP [292], a system for high-speed networks. It leverages "field-programmable gate array" hardware to convert packet streams into NetFlow for high-speed processing but due to the heterogeneous inputs required by each individual model further preprocessing of NetFlow data is a prerequisite. To assess the capabilities of CAMNEP towards latest cyberthreats, an evaluation is conducted in [34] using CTU-13 as recent botnet repository. Results bare few false alarms but also a rather low amount of discovered attacks. Other works with a focus on pure botnet detection are [215, 217, 293]. In [215] classic ML techniques are examined to detect botnets employing the ISOT data set among others. Considering a specific set of flow features, none of the postulated requirements is met. [293] captures NetFlow information to identify groups of endpoints that share similar communication patterns by applying clustering techniques and a linkage analysis. [217] exploits time intervals in addition to increase detection abilities. In neither of these cases, FS methods are examined explicitly to expose an indicator for most promising flow features. In fact, relative few effort centering around flow-based feature analysis can be found in literature. As one of the first works, [294] studies botnet traffic using different FS approaches to reduce dimensionality. In terms of predictive capabilities, classification results decrease only marginally on the reduced attributes while benefits are notable concerning runtimes compared to models relying on the entire feature set. Furthermore, the effectiveness of commonly used flow features is examined in [281]. Obtained results show rather poor performances using the ISCX-Botnet traces. A contrary view is reported in [295] where feature sets of different flow exporters are evaluated. The achieved output seems to be much better than exhibited in [281]. In [296], an attempt is made to extract meaningful features from the NSL-KDD data set. The authors further demonstrate to which extent these features are covered by IPFIX inferring a direct mapping between flows and TCP connections. Yet, this idealized assumption does not hold necessarily in practice as raised timeouts might split flows earlier (see Section 5.3). Moreover, the feature analysis is based on data broadly considered to be flawed (see Section 6.2). De-

spite its nonexistence at the time of conducting our original research in [5], we also want to mention [280] where a flow feature analysis is performed on a broader range of attack instances identifying different feature sets per attack type. Hence, that work is closely related but several points remain unclear. For instance, no details are provided about selected parameters for the employed FS algorithm or how data is partitioned and preprocessed. Other open points are to which extent these features sets generalize beyond the given data distribution using other benchmark data and whether those flow features are compliant with NetFlow/IPFIX given the self-developed flow exporting solution employed in that work. Lastly, we want to stress that the exposed feature sets rely on a FS approach, which rests upon a single inductive learner. This can be problematic as the features bias towards this learner (see Section 7.3.3). Other related works address the detection of brute-force and dictionary attacks. Their origins can be found in [9, 10]. Three common states reflecting SSH brute-force attacks are identified in [9] using hidden Markov models. This promising idea is implemented by the system SSHCure [297]. SSHCure is further improved in [11] by both attack and compromise detection. The work reported in [10] successfully determines characteristic patterns for SSH dictionary attacks but in contrast to [9] the model is developed based on a tree structure. Due to missing details about employed data, results are hard to reproduce. Other recent attempts countering brute-force attacks are given by [298, 299]. Further related concepts on flow-based intrusion detection are surveyed in [237, 238].

Among the presented effort in recent literature, most research on flow-based IDSs addresses specific problems, which constitutes point solutions. Only few activity is notable to detect most recent cyberthreats and classic attack types integrated into a holistic system. Particularly, flow feature analysis requires considerable attention not least because literature focus on botnet traffic predominantly. Hence, a general assessment towards other attack types has not been fully studied. In this chapter, we address this point by finding valuable flow features for various attack types mainly extractable from IPFIX data as an extension to [5]. Beyond the work in [5], we are not only including an isolated analysis of pure flow characteristics but also employ a richer set of features alongside with a broader number of learning algorithms that are examined.

7.3 Flow Feature Selection

This section is concerned with the analysis of flow features. We outlined employed traffic traces along with their basic characteristics first (Section 7.3.1). These are preprocessed to get a list of potential flow feature candidates (Section 7.3.2) and analyzed based on RST to finally obtain distilled features serving general intrusion detection using flow data (Section 7.3.3).

7.3.1 Employed Traffic Traces

Studying the general capabilities of flow features requires adequate data carrying a broad repertoire of different attacks within a fine-grained format. However, appropriate data sets with underlying GT can be rarely found. To mitigate these circumstances, we make

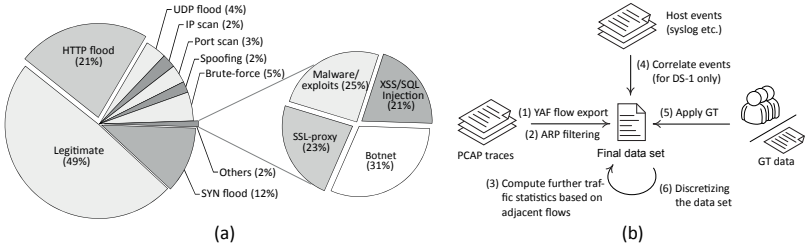

FIGURE 7.1: (a) Flow traffic distribution breakdown for DS-1 used to perform the feature analysis on various different attack types and (b) the steps required to obtain DS-1 and DS-2 for the FS process (adapted from [5])

use of the NDSec-1 traces introduced in Chapter 6. Recalling that NDSec-1 is designed as attack repository in the first place, it only caters few benign traffic by definition. This point raises the demand for ordinary background traffic as supplemental data source. We, therefore, strived for several hours of real legitimate traffic with the support of our research partner EDAG Engineering GmbH[2] (EDAG) by capturing raw packets from a floor distributing unit within EDAG's enterprise facilities in Fulda (Germany). These recordings contain conventional office traffic including frequently used protocols. In addition to this engagement, minor modifications are considered to both the legitimate part and attacks in order to obtain unbiased data for our objective. For instance, we remove very rare or uncommon data from the EDAG traces that were very specific to the automotive industry and focused on daily office occupations in an enterprise network instead. Moreover, several noisy attacks (e.g. HTTP and SYN flooding) inside NDSec-1 are undersampled with respect to a better attack distribution that otherwise would restrain the outcome of a learning algorithm. As a result of these actions, two data sources arise ready to be synthesized. Yet, their fusion is not a straightforward undertaking but can be managed pragmatically by the overlay methodology (see Section 6.1). In essence, it attempts to nest malicious activities into the benign network environment by mapping IP addresses and timings adequately. Fitting our purpose, this strategy is applied accordingly not least because of its relevance in literature (e.g. [271, 281, 300]). Completing the fusion process, a custom data set emerges with a certain balance between normal and attack captures. Obviously, this data distribution does not reflect a realistic network environment where legitimate traffic is expected to dominate malicious actions in the long run. Nonetheless, it encloses the characteristic footages of both traffic types that are required for a proper feature analysis on flow level, which is the focus of this chapter. We refer to this dataset as "DS-1". An overview of its data distribution is outlined in Figure 7.1 (a). Being able to further generalize the flow feature assessment, additional independent network captures need to be taken into account that fulfill similar properties compared to NDSec-1. With respect to the argumentation of Section 6.2, the only multi-purpose data set existing besides our original research had been ISCX-2012 which we continue to use herein. Despite its relative broad attack composition including flooding, brute-force or injection attacks, the intra-comparison of

[2] EDAG Engineering GmbH (see `https://edag.com`) is a company for product and plant development with a primary focus on the automotive industry. Its headquarter is located in Wiesbaden (Germany).

these harmful attacks falls short. This finding is particularly true when looking at them from a flow standpoint causing similar issues as mentioned above. Therefore, a fusion between ISCX-2012 and the widely accepted botnet captures CTU-13 is contemplated utilizing the same merging strategy as for DS-1, which yields a higher attack diversity from both an intra-perspective and inter-perspective. In terms of legitimate traffic, a high amount of HTTP data can be observed, which we soften carefully to get a balanced data distribution comprising 50% benign instances and 50% attacks. Out of simplicity, we call this data set "DS-2".

7.3.2 Feature Extraction and Preprocessing

Given the prepared network captures introduced in the previous Section 7.3.1, further preprocessing aspects need to be taken into account, which are detailed in this section. First, we discuss the suitability of native flow data and extract two new features that can be directly derived. Within this framework, we outline limitations and consider information residing in log messages and statistics across flows in addition forming four types of feature sets.

Recalling that biflows are considered in this work, flow features as given through Table A.1 can be extracted straightforwardly employing YAF as sophisticated IPFIX exporter with default timeout settings (see Chapter 5.3). However, we are not taking into account all of them for a deeper analysis due to a variety of motives. For instance, address-related attributes (see IE ID 8, 12, 27, 28, 56 and 80 of Table A.1) are not examined because this type of information is either anonymized for parts of the considered traces or tends to generate overfitted ML models. Other features that we try to circumvent directly are related to the port numbering (see IE ID 7 and 11 of Table A.1). They provide little contribution because of a number of dynamic strategies frequently utilized by attackers to evade detection (e.g. [281, 295]). As an exception, the application-level protocol or network service (e.g. DNS, HTTP, SSH) of a flow carries valuable information. The reason behind this vision relies on the observation that many attacks follow a fixed protocol structure that is an inherent characteristic requiring a certain amount of effort to manipulate an attack (see Section 5.2). Therefore, we attempt to extract this type of information using a three-step process. Initially, we exploit the existing but limited built-in DPI functionality of YAF to determine the underlying service, which, in our case, is based on the first 2048 bytes of payload that is captured per flow (see Section 5.3). If this fails due to a missing DPI implementation for the underlying protocol in charge, we refer to the IANA registry of "well-known ports"[3] to extract the service in a second step. Somehow, this second undertaking contradicts to the previous argumentation but as it is complementary to the first step we accept the usage of ports in this context. These determined services are used in the last step to map a flow to one of 11 defined "service categories" (SCs) eventually. The mapping between exposed network service and SC is depicted in Table 7.1. Early tests determining the SC with a representative sample of more than ten million flows reveal that 84.96% of the data can be classified with certainty using DPI while 8.55% are relatable by looking up the IANA list. In only 6.49% of the cases, no SC can be associated at all utilizing this procedure. Rectifying

[3] `https://iana.org/assignments/service-names-port-numbers/`

SC	Involved services
ARP	ARP
AUTH	LDAP, Kerberos
COMM	IRC, AOL, SMTP, POP3, IMAP, Telnet
DBMS	MSSQL, MYSQL, ORACLE
DNS	DNS
FTP	FTP, TFTP
HTTP	HTTP
HTTPS	HTTPS
MEDIA	RTP, RDP, VNC, SIP
MGMT	DHCP, SNMP, NTP, NETBIOS
SSH	SSH, SFTP

TABLE 7.1: Network service mapping to SC based on DPI and well-known ports

State	Description
SF	Expected behavior for TCP-based flows with handshake and tear-down between originating and responding party or the default state for flows related to UDP and ARP.
REJ	Connection attempt rejected by responder (e.g. due to a closed destination port).
S0	Connection attempt but no appropriate reply by responder.
S1	Connection established but no ordinary tear-down seen. Connection has not been terminated (also includes half-open connections).
S2	Connection established with an attempt to be closed by originator but no proper reply by responder.
S3	Connection established with an attempt to be closed by responder but no proper reply by originator.
RSTO	Connection established with an attempt to abort by originator.
RSTR	Connection established with an attempt to abort by responder.

TABLE 7.2: Estimated flow state to identify potential protocol violations

such situations, we define a pseudo category named "NoSC". Another modification to the standard flow exporting provided by YAF reflects the "state" of a flow. We do not mean the reason why a flow is terminated alone (see IE ID 136 of Table A.1) but if the communication reflected a corrupted flow. Inspired by the *conn_state* attribute[4] of Zeek, we are able to scrutinize flows that use TCP as transport protocol by employing four flag-related flow attributes and the end reason (see IE ID 14, 15, 136, 16398 and 16399 of Table A.1) delivered by YAF to estimate its state ultimately. For instance, we can figure out that a connection-oriented flow ended abnormally if a conventional three-way handshake was seen in the initial packets of the flow but a tear-down could not be perceived. Another example is the rejection of a connection attempt typically arising if the destination port is closed at the responder, which might point to a port scan. The entire semantics utilized to extract the flow state are outlined in Table 7.2. Finally, we would like to mention that YAF is not supporting the capturing of ARP information at all. Yet being able to analysis the amount of ARP spoofing and some comprised probings attacks (see Figure 7.1 (a)), further adjustments to the exporting are required. Therefore, we treat ARP packets similar to conventional UDP flows and extract them directly from the raw traces of DS-1 and DS-2.

[4] `https://docs.zeek.org/en/stable/scripts/base/protocols/conn/main.bro.html`

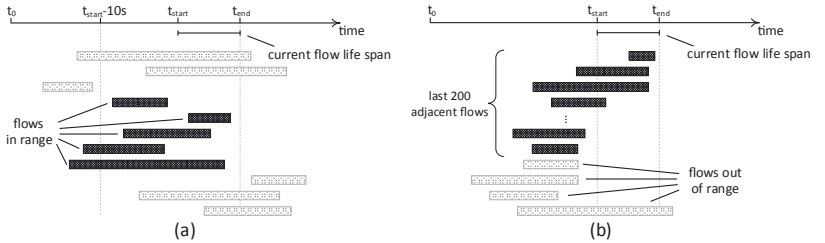

FIGURE 7.2: Capturing adjacent flows with different life spans around the current flow under consideration: (a) a time-based sliding window of size 10 seconds and (b) a conventional sliding window with a capacity of 200 flows

Beside these extensions to the customary IPFIX export of YAF, two preprocessing steps are considered in addition which we discuss briefly, i.e. the incorporation of "host events" and "traffic statistics". Intuitive reasons for these supplements are not only the limited visibility of flows because they represent aggregated network traffic activities leaving out details by purpose. Sometimes, information that belongs to the broader context of a network communication just remains at either side of involved endpoints and is simply out-of-sight from a flow perspective. Such meta data, however, can be extremely beneficial. Suppose we are given login attempts to a remote server (e.g. HTTP, SSH or RDP), then it is interesting to know the number of failed logins and put them into relation to all attempts performed recently to identify potential brute-force attacks. A similar argumentation is valid counting the number of failed requests to online resources of web servers or network shares, which might indicate an ongoing vulnerability scan, an SQLI or a malware spread in the network. Such information typically resides in local log messages and can be mapped to flow data if a causal relation between specific host events and network activities is inferred. This may lead to a better understanding of the overall network situation and, thus, provides deeper insights to security-related issues. Inspired by [33], we use timestamps and socket data to correlate both dedicated host events[5] and flows. Another limitation of available flow data can be exemplified when reviewing flow features alone. They are self-centered and just describe characteristics within a specific flow but no information about activities that surrounds it. Such a "neighborhood analysis" can be precious for the detection of rather noisy threats including DoS-like attacks. Therefore, we introduce two different mechanisms as an adaption from [22, 302]. These are window-based and outlined in Figure 7.2. On the one hand, Figure 7.2 (a) depicts the first approach that relies on a time-based window capturing flows that are within a range of ten seconds compared to the current flow under consideration. On the other hand, Figure 7.2 (b) employs a conventional sliding window of the last 200 adjacent flows. This way, we obtain two different views to compute several statistics creating a kind of situational awareness for a particular flow with surrounding information.

Ultimately, we apply the GT of each trace and discretized continuous features based on [303]. In that sense, discretization is not a requirement but it supports the feature

[5] To generate this type of information, we make use of a framework devised in [301] to collect and provide local log messages.

analysis and employed algorithms which we introduce and discuss in the next Section 7.3.3. Combined, we get a broader picture of the entire preprocessing, which is depicted in Figure 7.1 (b), while the list of all considered features is provided in Table 7.3. As a result, we can distinguish between three condition attribute types, i.e. features on flow level (F), host level (H) and those relying on window mechanisms (W). These in turn can be linked to four different "flow feature combinations" (FFCs) that are promoted further and represent the basic structure of concrete FVs under consideration:

- **F00:** FFC consisting of IPFIX-related data (see feature ID 1 to 33 in Table 7.3)
- **FH0:** F00 with associated host information (see feature ID 34 to 40 in Table 7.3)
- **F0W:** F00 enhanced with traffic statistics (see feature ID 41 to 56 in Table 7.3)
- **FHW:** all combined, i.e. F00 with host data and traffic characteristics

7.3.3 Algorithm and Results

As complexity of classification models rises with dimensionality, one of the main challenges in ML and related disciplines is to locate one or multiple reduced feature subsets with same or similar predictive abilities compared to the complete set of available condition attributes. Given the quantity of features at hand that we extracted previously (see Section 7.3.2), we are interested in such a FS due to three main motives: (i) it supports the elimination process of irrelevant and redundant information towards a condensed representation that is meaningful. (ii) By this means, fewer and more decisive information can be exploited countering overfitting largely, which is a desirable goal to detect intrusions based on flow data. (iii) Additionally, interpretability is a benefit that comes along with FS as a reduced feature set facilitates to study resulting models, which is explicitly interesting for models that are human readable such as DTs and rule sets (see Section 2.2.2). To this point, we only considered FS techniques from a specific angle reviewing Section 3.5 and 4.5 even though a significant amount of other methods have been introduced in literature over the last decades. To get a broader context, we, therefore, provide a brief overview along the three main existing categories of FS algorithms in the remainder of this paragraph. Proceeding this way serves as baseline to manifest our final design choice to propose a new algorithm eventually. The first of this three categories are "filter methods". Characteristically, they rely on statistical quality measures. Consistency-driven criteria (e.g. [144]), distance scores (e.g. [304]), correlation-based measures (e.g. [305]) or information-theoretic quantities (e.g. [306]) are most notable filter schemes to assess the merit of a certain feature. On this formal basis, they are transparent and clearly independent from any inductive method. As a consequence, filtering might be seen as preprocessing step right before the build process of a predictive model can commence. In contrast to this class of algorithms, the second category are "wrapper methods". They incorporate a learning algorithm and were broadly popularized through [307]. Using such techniques, the preselected learner is leveraged to measure the relevance of a particular feature by means of its predictive performance. Hence, feature sets produced by wrappers yield a higher predictive quality compared to filters due to their tight coupling with the learner. Yet, they might introduce more complexity in terms of runtime and certainly bias towards the chosen learner. Repre-

ID	Feature description	ID	Feature description	ID	Feature description
1	flow duration in milliseconds	20	payload length of the first non-empty pkt.	39	flip flops identified in ARP cache
2	transport protocol of the flow (e.g. TCP, UDP)	21	payload length of the first non-empty pkt. (rev. dir.)	40	# of modified files and events w. high severity at the responder
3	SC of the flow according to Table 7.1 or 'NoSC' otherwise	22	TCP flags of initial pkt.	41	# of flows to same dst. as current flow (window of past 200 adjacent flows)
4	# of pkts.	23	TCP flags of initial pkt. (rev. dir.)	42	% of flows to same dst. and SC as current flow (related to ID 41)
5	# of pkts. (rev. dir.)	24	union of TCP flags other than the initial pkt.	43	% of flows to same dst. as current flow w. state S0, S1, S2, S3 (related to ID 41)
6	# of octets in pkts.	25	union of TCP flags other than the initial pkt. (rev. dir.)	44	% of flows to same dst. as current flow w. state REJ, RSTO, RSTR (related to ID 41)
7	# of octets in pkts. (rev. dir.)	26	flow state according to Table 7.2	45	# of flows to same SC as current flow (window of past 200 adjacent flows)
8	total bytes in payload	27	average duration between pkts. in milliseconds (interarrival time)	46	% of flows to same SC and source port as current flow (related to ID 45)
9	total bytes in payload (rev. dir.)	28	average duration between pkts. in milliseconds (rev. dir.)	47	% of flows to same SC as current flow w. state S0, S1, S2, S3 (related to ID 45)
10	# of pkts. having between 1 and 60 bytes of payload	29	SD of ID 27 (first 10 pkts.)	48	% of flows to same SC as current flow w. state REJ, RSTO, RSTR (related to ID 45)
11	# of pkts. having between 1 and 60 bytes of payload (rev. dir.)	30	SD for ID 28 (first 10 pkts.) (rev. dir.)	49	# of flows to same dst. as current flow (window of past 10 seconds)
12	# of pkts. w. at least 220 bytes of payload	31	Shannon entropy calculation of the payload	50	% of flows to same dst. and SC as current flow (related to ID 49)
13	# of pkts. that contain at least 220 bytes of payload (rev. dir.)	32	Shannon entropy calculation of the payload (rev. dir.)	51	% of flows to same dst. as current flow w. state S0, S1, S2, S3 (related to ID 49)
14	# of pkts. that contain at least 1 byte of payload	33	flow end reason (e.g. idle, active, eof)	52	% of flows to same dst. as current flow w. state REJ, RSTO, RSTR (related to ID 49)
15	# of pkts. w. at least 1 byte of payload (rev. dir.)	34	# of all login attempts for SSH, Kerberos and RDP	53	# of flows to same SC as current flow (window of past 10 seconds)
16	the largest payload length of the flow	35	% of failed login attempts (related to ID 34)	54	% of flows to same SC and src. port as current flow (related to ID 53)
17	the largest payload length of the flow (rev. dir.)	36	% of login attempts w. elevated privileges (related to ID 34)	55	% of flows to same SC as current flow w. state S0, S1, S2, S3 (related to ID 53)
18	SD of payload length (first 10 non-empty pkts.)	37	# of all request attempts for web servers and network shares	56	% of flows to same SC as current flow w. state REJ, RSTO, RSTR (related to ID 53)
19	SD of payload length (first 10 non-empty pkts.) (rev. dir.)	38	% of failed request attempts (related to ID 37)	57	decision class

TABLE 7.3: Complete set of considered features

sentative wrapper approaches include SVM-RFE [308] or FSSEM [309] but essentially the combination of any inductive learner with an appropriate performance measure and search strategy constitutes a wrapper. The third group of FS algorithms are "embedded methods". They bring together the advantages of both filters and wrappers, i.e. a close integration of FS and learning process. Hence, approaches in that direction neither require a split to separate data into training and validation set nor depend on retraining phases for every feature subset as opposed to wrappers. The two most prominent families of embedded methods are decision trees such as ID3 [58] or variants of LASSO [310]. Besides these mentioned characteristics, one key issue affecting many FS methods is that input data are usually presumed clean. Such assumptions are critical in practice, which was already experienced in Section 4.5 where we elaborated on data imperfection and the rise of indispensable attributes that would be neglected otherwise. Hence, a framework is beneficial tightly connecting exploratory steps with FS functionality that is resilient to such situations. RST is a solid match to these postulated requirements not least because of its built-in analytics given through the indiscernibility relation in conjunction with the native support for data inconsistencies to identify proper attribute subsets. Especially, these points are essential when analyzing highly aggregated data such as IPFIX entities where fairly similar records may produce contrary decisions. These circumstances strengthen our motivation to utilize RST methodology to unveil distilled flow features at the very end. Given this argumentation, we propose a new FS algorithm based on RST in what follows. Furthermore, we report about practical results applying this method to given features on a specific data set.

Any of the two data sets (see DS-1 and DS-2) introduced in Section 7.3.1 in conjunction with the preprocessing of Section 7.3.2 can be considered as decision table $T_{\langle A,D\rangle}$ with $B \subseteq A$ and $E \subseteq D$ that may contain several reducts ultimately. Seeking all of them turns out to be a nontrivial task even for a moderate number of available features reviewing Section 3.5. This is one of the reasons why QuickReduct and our DB port QuickReductDB (see Algorithm 3.1 and 4.1) only attempt to find the first reduct that comes along. This greedy search strategy, however, can be wasteful particularly when several other reducts are located nearby an obtained one. To overcome this limitation, we suggest a new algorithm named BacktrackReductsDB depicted in Algorithm 7.1. In contrast to QuickReduct, it is capable to extract multiple reducts at once with minor computational overhead and takes advantage of our in-DB model employing Corollary 4.4 and 4.5. In essence, it pursues a forward selection similar to the hill climbing approach of QuickReduct but extracts the core $C \subseteq B$ as ideal starting point beforehand that is computed by procedure FindCoreDB given through Algorithm 7.2. Besides this systematic exploration of the feature space performed from the very start, BacktrackReductsDB further relaxes the strict "best-first strategy" of QuickReduct by expanding the search with all best solutions found in the current round. By introducing a working queue $Q \subseteq \mathcal{P}(B)$ initialized with C, this effectively means that we dequeue candidate sets $G \in Q$ progressively to determine $P \subseteq B \setminus G$ such that $G \cup \{a\}, \forall a \in P$ maximizes the positive region in the present iteration. If the feature configuration $G \cup \{a\}$ is a reduct, it is appended to the output R consequently or enqueued to Q otherwise. This routine progresses successively until Q is swept entirely. On the one hand, moving forward this way enables us to explore various alternative paths in the search space that are simply ignored by QuickReduct. On the other

Input: $T_{\langle A,D\rangle}, B \subseteq A, E \subseteq D$
Output: $R \subseteq \mathcal{P}(B)$

BEGIN
  $R, Q, C, G \leftarrow \emptyset$
  $C \leftarrow$ FindCoreDB$(T_{\langle A,D\rangle}, B, E)$
  IF C is reduct **THEN**
    $R \leftarrow C$
    RETURN
  END IF
  $Q \leftarrow C$
  LOOP
    $G \leftarrow$ first set in Q, $Q \leftarrow Q \setminus G$
    $P \leftarrow \arg\max_{a \in B \setminus G} \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{G \cup \{a\},E}(T))$
    IF $\max_{a \in B \setminus G} \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{G \cup \{a\},E}(T)) = \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B,E}(T))$ **THEN**
      $R \leftarrow R \cup (G \cup \{a\}), \forall a \in P : G \cup \{a\}$ is reduct
    ELSE
      $Q \leftarrow Q \cup (G \cup \{a\}), \forall a \in P$
    END IF
    EXIT WHEN $Q = \emptyset$
  END LOOP
END

Algorithm 7.1: Procedure BacktrackReductsDB to extract multiple reducts

Input: $T_{\langle A,D\rangle}, B \subseteq A, E \subseteq D$
Output: $C \subseteq B$

BEGIN
  $C \leftarrow \emptyset$
  FOR $a \in B$ **LOOP**
    IF $\mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B \setminus \{a\},E}(T)) \neq \mathcal{G}_{\{\mathtt{sum}(c_t)\}}(\mathcal{L}_{B,E}(T))$ **THEN**
      $C \leftarrow C \cup \{a\}$
    END IF
  END LOOP
END

Algorithm 7.2: Procedure FindCoreDB to extract indispensible attributes

hand, potential local optima can be overcome by "backtracking" previously identified configurations with same validity than the current path sought. Please note, further runtime considerations of BacktrackReductsDB are skipped at this point. Instead, we relegate interested readers to [186] where a formal elaboration is presented about this subject contrasting performances of QuickReduct and BacktrackReductsDB respectively.

For the purpose of exposing meaningful feature subsets covering a broad range of different attack types, we build upon DS-1 as primary data source. The reason behind this decision relies on a bigger variation of recorded intrusions and they turn out to be more competitive than those comprised in DS-2 which we experience particularly in Section 7.4. It should also be stated that due to the relative poor label details in DS-2 (see Table 6.2), DS-1 is stripped down to a binary classification setting to harmonize both data sets towards a better comparability of learning results later on. Moreover, we organize the FS process along two distinct FFCs, i.e. we perform the FS on (i) flow features only (see F00) and on (ii) both flow and traffic statistics (see F0W) in order to obtain different reduct versions of these FFCs comprising considerable fewer attributes. Using

FFC	Subset name	Feature IDs
F00	Core attributes	$1, 3, 6, 7, 10, 12, 16, 26$
	Reduct R_1	$1, 3, 4, 6, 7, 10, 11, 12, 14, 16, 17, 21, 26, 28$
	Reduct R_2	$1, 3, 6, 7, 10, 12, 13, 14, 15, 16, 17, 20, 26, 27$
	Reduct R_3	$1, 3, 5, 6, 7, 8, 10, 12, 14, 15, 16, 26, 27, 28$
F0W	Core attributes	$3, 6, 22, 41, 42, 44, 45, 46, 49, 50, 53$
	Reduct R_1	$1, 2, 3, 6, 7, 22, 24, 26, 41, 42, 44, 45, 46, 49, 50, 53$
	Reduct R_2	$1, 2, 3, 6, 7, 22, 23, 24, 41, 42, 44, 45, 46, 49, 50, 53$
	Reduct R_3	$1, 2, 3, 4, 6, 7, 22, 24, 41, 42, 44, 45, 46, 49, 50, 53$

TABLE 7.4: Exposed reducts and core attributes per FFC using DS-1

this differentiation supports us to study the predictive performances among the two FFCs in more detail, which is not offered in [5]. The other variants FH0 and FHW are not considered for the FS process recalling that host data are treated as supplemental information only. Having mentioned these points, we finally turn over to the practical application of BACKTRACKREDUCTSDB on the entire set of condition attributes given through F00 and F0W. Obtained results yield three distinct reducts per FFC outlined in Table 7.4. The intra-comparison of F00 reveals that all reducts share the same cardinality, i.e. a size of 14 features. Yet, they are quite different when contrasting them pairwise. While R_1 and R_2 as well as R_1 and R_3 have only ten attributes in common (i.e. 71.43%), the greatest homogeneity is provided by R_2 and R_3 with a similarity of 78.57%. This outcome is supported by the core of these reducts only containing eight features (i.e. 57.14%). On the contrary, the comparison of F0W demonstrates different insights. All three reducts of that variant embody two more attributes than reducts from F00 with a consistent likeness of 15 features (i.e. 93.75%), while the core of that assessed FFC entails 11 attributes (i.e. 68.75%) for all three reducts. Further important points can be exhibited when performing an intra-comparison of both variants. In this respect, all reducts of F0W follow the same structure. They are based on eight flow features and encompass eight traffic attributes. Putting this result into perspective with F00 indicates that certainly more information is comprised using derived traffic statistics rather than data natively supplied by YAF as BACKTRACKREDUCTSDB equally selects 50% of both feature types. This hypothesis is also confirmed when reviewing the positive region of the respective reducts where F00 only holds 91.42% of flows and reducts relying on F0W include 97.84% of the data. Ultimately, we would like to stress compression aspects in contrast to Table 7.3. Out of 33 available dimensions for F00, extracted reducts only contain 14 features which is a 2.36-fold reduction with same expressiveness. Even more impressive is the dimensional compression for F0W where only 32.65% of all original attributes are required to produce an equivalent classification, i.e. a 3.06-fold decrease. Despite of this successful reduction, further practical assessment is needed to confirm the outcome of these reducts which is addressed next in Section 7.4.

7.4 Empirical Assessment

From a rough set standpoint, all exposed reducts (see R_1 to R_3) in Section 7.3.3 share the same predictive capabilities for either variant F00 and F0W. Despite this formal consideration, conventional ML-based algorithms might react quite differently due to their

varying induction strategies. Thus, an important point is to examine the effectiveness of these feature sets in practice, which is the undertaking of this section. Therefore, we introduce several existing supervised learners and group them into five distinct categorizes. Furthermore, important performance measures are outlined to estimate individual classification performances (Section 7.4.1). As these learners enclose distinct parameter settings, different combinations of them have to be studied consequently. Thus, a common strategy is employed to figure out the best settings to obtain high predictive qualities discerning attack and legitimate data. This task is carried out using DS-1 as testbed (Section 7.4.2). The analysis is further extended along two dimensions. First, we incorporate runtime aspects to find the most efficient models per category. In the second stage, an attempt is made to generalize received results by applying the most promising learners to DS-2. Note, we also highlight results incorporating host events. Yet, this examination can only be conducted for DS-1 because DS-2 comprises network data only (Section 7.4.3). Finally results are recomposed and discussed (Section 7.4.4).

7.4.1 Learner Categories and Measures

In order to obtain a broader overview of the reduct potentials for both FFCs, a large set of learners has to be considered eventually implying a unified test environment that needs to be setup in addition. The statistical computing software "R" [311] is selected for this purpose including the libraries "caret" [312] and "RWeka" [99, 313]. Using this platform as baseline is very encouraging because it provides a large arsenal of distinct ML implementations from which we can pick and choose. Out of this collection, we carefully assemble a compilation of 15 representative methods. Note that only conventional base learners are considered at this stage rather than advanced learning techniques which we defer to Chapter 8 and beyond. These selected base methods can be roughly organized around five different classes of algorithms which we call "learner categories" (LCs). In this regard, the first category only comprises approaches that construct tree structures as final model. It is named "TREE" for this reason, which consists of the well-known DT C4.5 and ID3. Ctree [314] as additional tree learner is attached to this LC as well. On the contrary, the second group takes into account decision rules rather than DTs. This group is called "RULE" for the sake of brevity. It contains C5.0-R holding a transformed rule set from the DT C5.0[6]. The second and third learner in this LC are PART [315] and RIPPER [316]. The first two learners of category "ANN" are MLP and MLP-WD. In essence, they pose regular multilayer perceptrons (e.g. [114, 317]) whereas MLP-WD also incorporates a level of decay. The last learner in this category is RBFN representing a radial basis function network [318]. "SVM" is the fourth LC and comprises the three SVM-based learners SVM-L, SVM-P and SVM-R. All three use different kernel methods to induce distinct theories, i.e. a linear, polynomial and radial basis function kernel in consecutive order. The final category encloses a heterogeneous list of algorithms and is, therefore, named "MISC". The first learner encompassed by MISC is HDDA and rests upon high dimensional discriminant analysis [319], while the second is kNN representing the prominent k-nearest neighbors algorithm. The last approach of this group is called NBay and implements naive Bayes

[6] `https://rulequest.com/see5-info.html`

		Prediction	
		malicious	**benign**
GT	**malicious**	TP	FN
	benign	FP	TN

TABLE 7.5: Prototypical confusion matrix in the context of intrusion detection that summarizes the result of a model on a binary classification problem

(NB) estimating the most likely class based on the posterior probability (see (2.1)) assuming feature independence given the class. With these LCs and involved learners in place, we are interested in their classification performance when confronted with DS-1 and DS-2. Such an examination not only provides insights on how meaningful our proposed reducts are, but also enables comparisons among classifiers inside a single group or across multiple ones. Yet, appropriate benchmark measures have to be defined beforehand. On that account, let us suppose a conventional intrusion detection scenario distinguishing between flows that are tagged as "benign" and those that are labeled as "malicious". In this binary classification setting, we are able to summarize the predictive capabilities of a model using a 2×2 "confusion matrix". Such a tableau is given through Table 7.5 where "true positives" (TP) reflect attack flows correctly hit, "false negatives" (FN) express missed attacks, "true negatives" (TN) indicate benign flows correctly predicted and "false positives" (FP) outline false alarms. Based on these absolute counts obtained during evaluation, we are able to extract three important rates that are commonly used in intrusion detection analyses (e.g. [44, 302, 320]), i.e. the "true positive rate" (TPR), the "false positive rate" (FPR) and the "classification accuracy" (ACC). In that sense, the TPR highlights the percentage of detected attacks over all given threats and can be calculated by

$$TPR = \frac{TP}{TP + FN} \tag{7.1}$$

whereas the FPR underlines the amount of false alarms in relation to all unintentional flows computed by

$$FPR = \frac{FP}{TN + FP} \,. \tag{7.2}$$

Finally, ACC compresses the entire classification result by relating correct predictions to all examples seen. Consequently, this measure can be expressed by

$$ACC = \frac{TP + TN}{TP + FN + TN + FP} \,. \tag{7.3}$$

Some synonyms for the TPR are "hit rate" or "detection rate" while the FPR is also named "false alarm rate". For the sake of completeness, we would also like to state three supplemental measures that can be directly derived from them, i.e. the "false negative rate" defined by $1 - TPR$ (also called "miss rate"), the "true negative rate" determined by $1 - FPR$ and the "error rate" calculated by $1 - ACC$. In what follows, we use this terminology in conjunction with a standard five-fold cross-validation (e.g [66, 99, 321]) to estimate a model's fitness to the underlying problem domain exemplified through DS-1 and DS-2. However seeking for the best learner and settings per LC, DS-1 is used exclusively in the next Section 7.4.2 following the argumentation of Section 7.3.3.

7.4.2 Parameter Selection Using ROC Analysis

Being able to explore and showcase obtained classification results of each individual learner with varying parameter settings during our experiments, we leverage the "receiver operating characteristic" (ROC) space (e.g. [65, 322]). Conventionally, one data point $\langle x, y \rangle$ in that two-dimensional space represents the performance of a single model with a fixed parameter setting where x is the FPR and y is the TPR. This way, different parameter combinations affecting TP and FP can be pictured at the same time. Obviously, the most optimal performance is reached at $\langle 0, 100 \rangle$ whereas $\langle 100, 0 \rangle$ is the weakest, i.e. the "ROC heaven" and "ROC hell" respectively. Points along the diagonal from $\langle 0, 0 \rangle$ to $\langle 100, 100 \rangle$ signify random guessing. Thus, one is interested in produced classification results $\langle x, y \rangle$ where $y \gg x$ that are close to the ROC heaven[7]. Once a set of data points is collected, finding the best theory is straightforward. We simply have to compute the distance of each point to the most ideal and select the lowest. Formally, this can be expressed by

$$\underset{\langle x,y \rangle \in \mathbb{R}^2}{\arg\min}\ dist_{opt}(\langle x, y \rangle) \tag{7.4}$$

where

$$dist_{opt}(\langle x, y \rangle) = \sqrt{x^2 + (y - 100)^2} \tag{7.5}$$

is the "Euclidean distance" to the ROC heaven. Such points are potentially optimal and are located on the "ROC convex hull" (ROCCH), i.e. the convex hull of all considered points in ROC space [322]. All other points below the ROCCH are suboptimal and can be discarded. Note, we make a minor modification to this common analysis in order to maintain a compact representation over our large set of experiments. Instead of outlining the performance of each reduct and model as separate point, we average the obtained classification results per model and FFC receiving a condensed description of the reduct performances in ROC space. Starting with the tree-based classifiers, the application of reducts R_1 to R_3 for F00 reveals that the best performance can be attributed to C4.5. It produces an average TPR of 94.12% followed by ctree, which is very close with 93.76%. ID3 is in third position catering for 88.05%. Despite these relative close results, a bigger deficiency can be identified when it comes to FP. Both ctree and ID3 carry a FPR that is at least 3.43 times higher than the outcome of C4.5. In this regards, ctree shows the poorest performance where almost every second legitimate flow is falsely classified as malicious. This output manifests that both learners cannot compete with C4.5 using this FFC, which is supported by their varying distances to $\langle 0, 100 \rangle$. $dist_{opt}$ of C4.5 is only 7.55, whereas the corresponding smallest distances of ID3 and ctree are 20.15 and 42.03 accordingly. Better results can be demonstrated taking into account those reducts of F0W. On these extended feature sets, ctree and ID3 decline their distance to C4.5 in relation to raised false alarms but they are still behind overall. The best version of C4.5 in our tests reaches a TPR of 97.82% and a FPR of 3.44% on average, which is 3.70% higher and 1.28% lower compared to the assessment of F00. The same tendency is observable for ID3. It improves by 4.53% and 4.23% concerning its TPR and FPR, while ctree's hit rate drops massively by 18.74%. Note that this loss in performance cannot be compensated by the large decrease of its FPR, which is 5.35 times lower in

[7] Strictly speaking, it might also be a desirable objective to aim for points nearby $\langle 100, 0 \rangle$ because we simply can invert a theory's output turning it into an effective one.

relation to its F00 setting. This outcome clearly makes ctree undesirable from a practical standpoint. Turning over to the combined results on category RULE using F00, the best output is achieved using PART followed by C5.0-R and RIPPER. All of them receive a TPR nearby 93.00% and even the FPR of PART and C5.0-R are close with 4.94% and 4.63%. However, RIPPER drops behind with a FPR twice as high. With respect to F0W, results slightly differ. While PART is still ahead performance-wise, C5.0-R only can improve marginally compared to the F00 setting such that it is outpaced by RIPPER due to RIPPER's solid improvement in ROC space. Furthermore, a large increase in performance can be relegated to the ANN category contrasting F00 and F0W where the hit rate of all three learners raises by at least 6.76% and the FPR reduces by at least 1.71% using the introduced traffic features in addition. In this regards, the best model for F00 is obtained by RBFN with a TPR of 89.77% and a FPR of 7.48%. Likewise, MLP produces 96.53% hits and 5.77% false alarms using F0W. Considering the SVM category, SVM-R provides the best outcome with an average TPR of 93.46% and a FPR of 8.74% applying R_1 to R_3 of variant F00, which is right in front of SVM-P and SVM-L. Yet, their performance is not far off except for the FPR that is at least 5.53% higher. Rank-wise, the same results can be manifested for R_1 to R_3 on F0W but SVM-P and SVM-L show a better outcome in terms of FP. Finally, we state results from MISC. For F00 and F0W, the best model is delivered by NBay. It acquires a TPR of 94.04% using pure FVs, which can be extended to 96.93% applying F0W. The only learner that can compete in this regard is kNN showcasing a marginally lower hit rate of 92.39% at F00 and 96.43% at F0W. Similar numbers can be reviewed facing raised false alarms. Both produce very close FPRs nearby 4.50% using F00 and 2.10% employing F0W. Their similarity is further emphasized by comparing their close distances to the optimum with a difference of 1.54 for F00 and 0.39 for F0W. HDDA is far away from these solid results due to its weak FPR that is at least 20.83% for both FFCs. Combining all of these findings documents that NBay produces the best model on F00 followed by C4.5 and PART, which are very close in terms of their TPRs ($> 92.89\%$) and FPRs ($< 4.94\%$). The best model of category ANN, i.e. RBFN, can only compete with them in terms of hits but the amount of false alarms is 2.54% higher than the weakest result of the top three. SVM-R as best model for category SVM is even behind RBFN. Turning to F0W, the ranking of the three best models only changes sparsely as opposed to F00, i.e. NBay remains in first position followed by PART and C4.5, whose TPRs and FPRs improve to at least 96.92% and 3.44% respectively. SVM-R and MLP are in fourth and fifth position. A graphical representation of the entire ROC analysis with different parameter combinations per learner is outlined in Figure 7.3, while the best parameter settings are detailed in Appendix A.2.

7.4.3 Runtime Incorporation and Consolidation

Based on these results, we were able to find the best performing learners and parameters per LC and FFC so far. However, this methodology does not incorporate time demands each learner comprises in addition as high runtimes can become critical in practice. Hence, we make an attempt to include the training time and the processing time as supplemental dimension stretching the model selection process further. Therefore, we reuse the best model performance per LC (see bold data points in Figure 7.3) as most

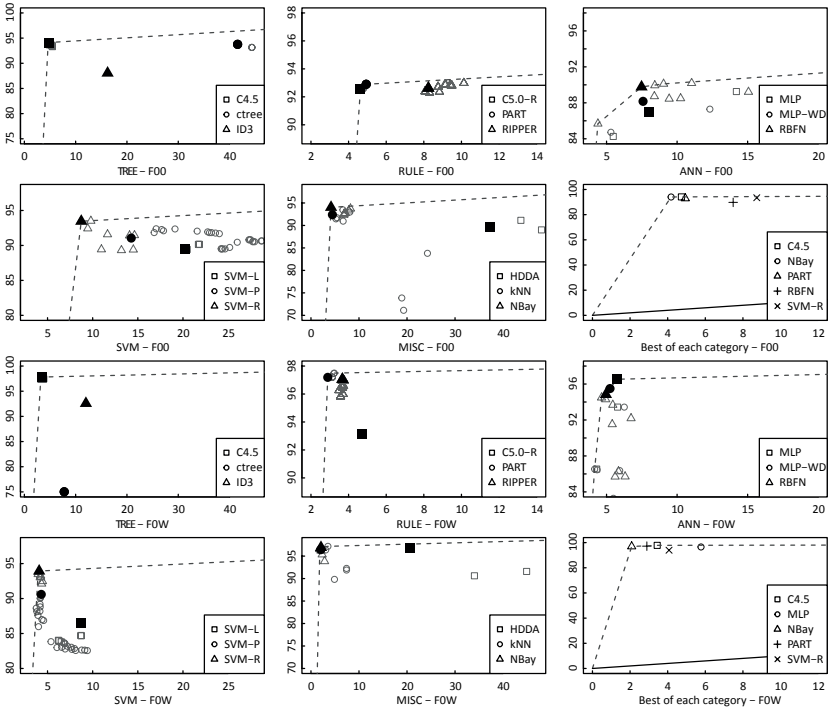

FIGURE 7.3: Parameter selection in ROC space for F00 and F0W on each learner per LC; each point represents the average classification performance over all three exposed reducts constituting distinct parameter setting; bold points show the best parameter setting per learner; dashed and solid lines highlight the ROCCH considering all classification results and performances by chance respectively; TPRs and FPRs are expressed in percent

promising candidates and record their average timing based on five repetitions. These numbers are obtained on a selected testbed[8] inside a dedicated virtualization platform[9] employing a single virtual CPU (vCPU) only. To assess these results, we combine them with the classification performances received earlier and depict them for each reduct to establish a rank-wise comparison. We denote that ACC is used instead of FPR and TPR this time to highlight the classification output[10] per reduct in a more aggregated form. The outcome of this supplemental examination is illustrated in Table 7.6. Obviously, the ranks in terms of ACC clearly reflect the tendencies of the ROC

[8] Microsoft Windows Server 6.3 (std., build 9600), R 3.5.2 (x64), caret 6.0-84, RWeka 3.9.3, JDK 1.8.0.51, 8× CPUs, 16 Gbyte RAM, 256 Gbyte HDD

[9] VMWare ESXi 6.5.0 (build 4564106) on a Fujitsu Primergy RX200 S7: 2× Intel Xeon CPU E5-2630 (2.30 GHz), 96 Gbyte RAM, 4× 1 Tbyte SAS HDD configured as RAID-5

[10] Using ACC can distort classification results considerably which we further elaborate in Chapter 9. Yet, DS-1 and DS-2 represent balanced class distributions (see Section 7.3.1) and so ACC can be used without greater concern in this section.

FFC	LC	Learner	*ACC*				Timing			Mean
			R_1	R_2	R_3	Rank	Train	Proc	Rank	rank
F00	TREE	ctree	76.18	75.78	76.81	3.00	2.20	**1.26**	**1.50**	2.25
		C4.5	**95.00**	**94.47**	**94.61**	**1.00**	7.77	1.62	2.50	**1.75**
		ID3	87.19	84.58	84.92	2.00	**1.42**	1.65	2.00	2.00
	RULE	C5.0-R	94.17	**93.83**	93.89	1.67	**15.19**	10.91	**2.00**	1.84
		PART	**94.26**	93.63	**94.03**	**1.33**	22.16	1.96	**2.00**	**1.67**
		RIPPER	92.85	92.14	91.60	3.00	48.54	**1.69**	**2.00**	2.50
	ANN	MLP	92.78	86.61	89.03	3.00	**21.05**	4.58	**2.00**	2.50
		MLP-WD	93.19	88.44	89.18	2.00	21.34	3.17	**2.00**	2.00
		RBFN	**93.44**	**89.04**	**90.90**	**1.00**	21.70	**2.95**	**2.00**	**1.50**
	SVM	SVM-L	85.20	84.17	84.52	3.00	**11.33**	**3.09**	**1.00**	**2.00**
		SVM-P	90.45	86.69	88.10	2.00	37.80	10.88	2.00	**2.00**
		SVM-R	**92.37**	**91.67**	**90.23**	**1.00**	60.85	14.16	3.00	**2.00**
	MISC	HDDA	81.56	72.02	75.40	3.00	6.93	**2.86**	2.00	2.50
		*k*NN	93.06	90.85	91.55	2.00	4.94	19.26	2.50	2.25
		NBay	**95.60**	**94.40**	**94.70**	**1.00**	**1.71**	2.91	**1.50**	**1.25**
F0W	TREE	ctree	84.14	83.75	83.77	3.00	6.89	2.36	2.50	2.75
		C4.5	**97.24**	**97.16**	**97.11**	**1.00**	10.04	2.29	2.50	1.75
		ID3	91.15	89.55	90.20	2.00	**2.43**	**1.96**	**1.00**	**1.50**
	RULE	C5.0-R	94.45	92.33	95.95	3.00	**17.83**	12.81	2.00	2.50
		PART	**97.29**	**97.07**	**97.08**	**1.00**	23.60	**2.09**	**1.50**	**1.25**
		RIPPER	96.76	96.62	96.66	2.00	51.33	2.21	2.50	2.25
	ANN	MLP	**95.44**	**95.41**	**95.18**	**1.00**	**22.45**	4.89	**1.50**	**1.25**
		MLP-WD	95.12	95.37	94.85	2.00	25.17	4.92	3.00	2.50
		RBFN	94.92	95.25	94.68	3.00	23.09	**3.13**	**1.50**	2.25
	SVM	SVM-L	90.95	87.93	87.96	3.00	**12.58**	**3.43**	**1.00**	**2.00**
		SVM-P	93.36	93.13	93.17	2.00	40.09	13.98	2.00	**2.00**
		SVM-R	**96.10**	**94.95**	**93.45**	**1.00**	65.15	19.77	3.00	**2.00**
	MISC	HDDA	88.45	87.46	87.62	3.00	10.33	4.26	2.50	2.75
		*k*NN	96.03	94.91	94.49	2.00	8.75	34.11	2.50	2.25
		NBay	**97.36**	**97.20**	**97.50**	**1.00**	2.82	3.85	**1.00**	**1.00**

TABLE 7.6: Classification performance on DS-1 per FFC and LC incorporating time demands as relevant dimension; ranks refer to FFC and category respectively; *ACC* of R_1 to R_3 are in percent, while the averaged training time (Train) and processing time (Proc) are in seconds; bold numbers indicate best results

results. Incorporating time demands, though, this performance can only be confirmed partially. While for category ANN, RULE and MISC trends are still the same, TREE and SVM showcase different results. This is particularly true for F0W on TREE where ID3 reaches a better overall rank of 1.50 than C4.5 with 1.75 as it requires fewer time on training and processing the respective input. But for all that, a significant difference can only be documented during training as their processing mismatch is marginal. Including this finding yields a tie between both models in total. Turning to category SVM on both FFCs, the dominant classification results of SVM-R are still apparent, which come at the cost of higher training and processing times as opposed to SVM-L and SVM-P. Thus, all three models obtain a rank of 2.00 per FFC constituting a tie overall.

To further extent our practical analysis, we state the results of the most auspicious learners uncovered earlier and apply them on DS-2 with the same parameter settings as for DS-1. We also include those classifiers where a tie could be detected or those that are at most 0.25 rank points away from the best model per category being able to illuminate also close results. The outcome is highlighted in Figure 7.4 using boxplots.

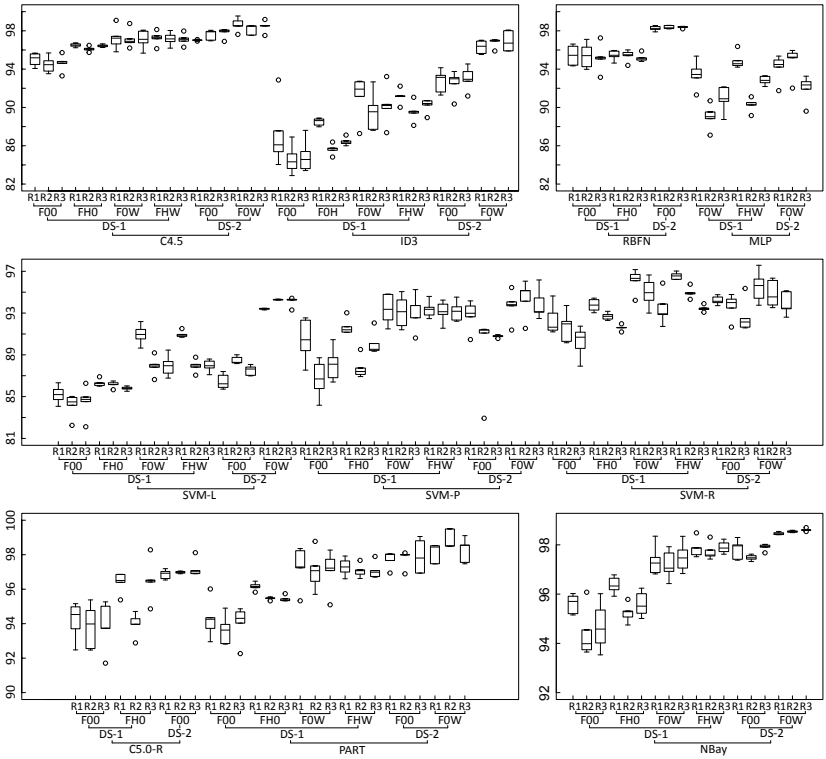

FIGURE 7.4: *ACC* over the cross-validation as boxplots on DS-1 and DS-2 using the most promising models per reduct, FFC and category; *ACC* is expressed in percent

Note that obtained numbers from DS-1 are incorporated for reference purpose as well as resulting performances when correlating host information (see feature ID 34 to 40) to the exposed reducts of F00 and F0W, i.e. FH0 and FHW. A closer look to the data shows that similar tendencies to prior results can be uncovered. On F0W, *ACC* is better by 2.41% on average as opposed to results of F00 applying DS-2. An exception is the leap obtained when running SVM-L on F00 and F0W producing an improved *ACC* of 6.45% for all three reducts on average. SVM-P and SVM-R are not sharing these specifics, which is probably the reason why even SVM-L slightly outperforms the other two SVMs on R_3 for variant F0W. However, comparing their classification output solely on F00 demonstrates a poorer outcome for SVM-L which is 3.59% lower than SVM-P. The gap to SVM-R is 6.02%. Let us also briefly summarize the direct comparison between C4.5 and ID3 as well as contrast C5.0-R and PART on DS-2. With regards to these confrontations, we can clearly see that C4.5 still outpaces ID3 using all three reducts per variant. On F00, the *ACC* diverges by 4.71% on average while a difference of 1.74% can be observed for F0W. The results for C5.0-R and PART on F00 are even closer where PART is only 0.80% better on average. Note, we are not stating the result between both

models on F0W as a clear winner could already be identified on DS-1 through Table 7.6. Conflating the results of DS-1 and DS-2, we generally can say that the performances on DS-2 is very similar or even better than those on DS-1. Yet, the tendencies along the reducts are not the same necessarily. For instance, let us take a look at the output of MLP using F0W where on DS-1 the best classification is performed by R_1 followed by R_3 and R_2. This order is not given when applied to DS-2 because we have R_2, R_1 and R_3 performing best to worst. Similar effects are particularly notable for SVM-L on F00 and F0W and for PART on F0W. Turning over to the results when enriching FVs with host information, appealing implications could be recorded. Recalling that the outcome can only be reported for DS-1, classification performances are further increased for both FH0 and FHW in comparison to F00 and F0W. On average, *ACC* can be boosted by 1.43% incorporating host events while numbers on FHW only increase by 0.08% on average compared to F0W. However, the impact for SVM-R and NBay is slightly higher where the *ACC* of those particular models raises 0.32%. The final finding we want to stress at this point refers to the enhancement of flows with host events. Their correlation not only produces equally good or better results in terms of *ACC* but it also provides a reduction in terms of variance, which is certainly a desired outcome.

7.4.4 Discussion

In this Section 7.4, we examined different FFCs towards their reliable detection of classic as well as state-of-the-art network intrusions. Against this background, we could unveil that obtained reducts based on flow features exclusively (see F00) provide tempting results not only from a RST perspective. In fact, experiments with several classes of ML algorithms demonstrated an *ACC* between 84.17% and 97.91% for the best models separating malicious from benign network activities. For the most part, this can be rated a solid outcome in light of these very basic learners. To amplify performances, we also incorporated traffic features to straight FVs resulting in a new set of reducts (see F0W). This enhancement further improved classification results by 3.02% on average for the respective models. In no observed case, predictive qualities were degraded. Thus, we rate this additional source of information as valuable complement due to insights that are not contained in pure IPFIX entities. Another attempt was carried out by correlating host events to existing FVs (see FH0 and FHW) as supplemental input in order to study their impact on the overall classification. Results illustrated that their effect is most notable on FH0 with an *ACC* that increased by 1.43% on average. The performance gain for FHW was only marginal suggesting that the effect of host events only plays a minor role due to an already clear distinction of most attack and normal footprints paved by flow and traffic features. With the support of host information, yet, a lower variance could be conducted throughout our experiments such that trained inducers produced more consistent results in our cross-validation setting. Most of these discussed outcomes could be obtained on both employed data sets DS-1 and DS-2 with fixed parameter settings. Given their independence further stretches the meaning of our empirical assessment confirming that the proposed feature sets indeed support intrusion detection based on a reduced amount of available flow and derived information. In this respect, a further finding is that performances of the selected learners on DS-2 are very similar or even better than on DS-1. Only in two out of 51 cases, the outcome on DS-1 was slightly

higher. Thus, it can be concluded that DS-1 is a very competitive data set with several attack instances that are difficult to identify in comparison to state-of-the-art intrusion detection data sets represented by DS-2. This further acknowledges our choice of using DS-1 as primary source for both the FS process and the ascertainment of auspicious base learners. Despite these discussed results, there is still room for improvement particularly when considering further operational aspects where potential changes of the underlying data generating process can destabilize the performance of an IDS very easily. Another point is an imbalance in the class distribution. This is a typical issue in our problem domain where attacks are rarely observed in comparison to legitimate traffic. This aspect has a certain impact to involved detectors and selected performance measures respectively.

7.5 Summary

With the proliferation of NetFlow/IPFIX technology that is supported by many packet forwarding devices nowadays, monitoring QoS aspects of the underlying network infrastructure becomes scalable and affordable at the same time. These points make flow technology also very attractive from a security standpoint. In fact, intrusion detection based on flows is becoming increasingly popular having a very active research community. Anyhow, existing solutions concentrate on specific problems raising the questions to which extent flow technology can be exploited to build general-purpose flow-based IDSs. This in turn implies the utilization of a general set of features extractable from flow data, but elaborated studies for meaningful features focus mainly on botnet detection.

To address this gap, this chapter concentrated on an analysis of flow features towards the coverage of a broader set of cyberthreats, which can be considered one of the first contributions in this direction. Therefore, two benchmark data sets were assembled comprising classic as well as recent attack footages constituting the data sources of this analysis. We nominated 56 feature candidates in total that potentially could be valuable for a reliable intrusion detection process using flows as baseline. To seek for meaningful features, we employed RST that is well-suited for problem domains suffering from uncertainty. As a result of this activity, several condensed feature subsets could be obtained reducing dimensionality largely. Distinguished by their FFC, all disclosed feature subsets shared the same predictive capability from a RST perspective. However, these formal considerations can vary in practice due to different learning strategies employed by ML algorithms. Thus, we further examined the credibility of all feature subsets by selecting a list of 15 distinct learners. These were applied to the exposed feature sets on both compiled benchmark data sets DS-1 and DS-2. Considering the very basic nature of these utilized inducers, a broad repertoire of different attacks could already be unveiled using pure flow features but predictive performances could even be boosted when incorporating security-related log information collected at participating endpoints or traffic statistics across adjacent flows. Yet, the performance gain combining both supplements was marginal suggesting to use only one of these sources apart from pure flow features.

Since, the focus of this chapter was on the analysis of flow features whether being capable to separate both benign and malicious traffic footprints adequately, important aspects were neglected that potentially could become pitfalls in practice. These include concept drift and class imbalance as two serious problems affecting our target domain. On the one hand, concept drifts can occur if the underlying data distribution is subject to change caused either by an attacker or by components of the network infrastructure. On the other hand, class imbalance is problematic as many learning algorithms are influenced by skewed class distributions typically biasing towards the majority class. As such, appropriated amendment is required which we address in the following Chapters 8 and 9. Another important aspect is the extra effort required to either compute traffic statistics or to successfully correlate host information to existing flows. This subject is deferred to Chapter 10 and elaborated as part of practical tests.

Chapter 8

Anomaly Detection

Anomaly detection engines are a key ingredient to uncover new unseen attacks or zero-day exploits. This essential property stands out from other frequently employed mechanisms in network security, which is the underlying motivation why anomaly detection plays also a central role for the HFIDS development in this work. Despite this tempting feature, existing techniques inherently produce very high false alarm rates posing a critical factor in terms of operational aspects. This issue is further increased reviewing our complex problem domain where concept drift is apparent and the availability of GT data is limited such that only unlabeled data can be expected after a certain point in time. Addressing these challenging circumstances, our focus is on the conceptual design of ADC in this chapter consisting of two parts, i.e. a supervised and an unsupervised component. The supervised part is responsible to detect existing attacks and mutations reliably whereas the unsupervised part takes care of new significant changes in the data including novel attacks or legitimate modifications of the network. First experiments on an attack evasion scenario unveil that the supervised part of ADC is capable to successfully anticipate the invader's intention. In more radical situations where the attack frontier changes abruptly or the underlying network environment is altered gradually, the unsupervised part of ADC is required. Results in both settings show that ADC performs better than various single classifiers or a static composition of them.

8.1 Introduction

In the previous Chapter 7, we experienced that the detection of intrusions combining flows and ML basically works even though obtained predictive performances fluctuated dependent on the utilized learning algorithm and FFC. Nevertheless, the underlying setup was idealized so far. We neglected important aspects that are crucial for the overall HFIDS development, namely, anomaly detection, class imbalance and nonstationary environments. In this chapter, we deal with all three points by collating the basic conceptual design of ADC. We start in this section with a brief discussion on anomalies followed by an outline of ML-based techniques to identify anomalies guiding our final design choice for ADC. In this respect, another closely related factor is the learning environment in which ADC operates. We point out the problem of limited GT and review solutions employed by other authors to finally tie down a reasonable setup being able to train and run ADC in a realistic environment. In the remainder of this chapter, we motivate the utilization of a committee of models for the classification part of ADC instead of using a single theory and present important background (Section 8.2). Based on that, the design of ADC is outlined in detail. Apart from its internals, an experiment is conduced to exemplify its benefits over conventional techniques followed by a discussion (Section 8.3) and a closing summary (Section 8.4).

So far, we mentioned anomaly detection systems several times but never came down to the point what an anomaly actually is. Therefore, let us briefly review earlier works to reconcile their view on anomalies with our vision. According to the landmark paper of anomaly detection in the context of intrusions detection given in [43], an anomaly corresponds to an abnormal behavior that is observed in a system. While this definition can be understand intuitively permitting some room for interpretation, authors of other highly influential works in the field, yet, explicitly assume that anomalies are infrequent and rather exceptional events (e.g. [288, 323, 324]). This refined view to anomalies complies with the general observation surveyed in [13] stating that literature oftentimes use the term “anomaly” and “outlier” interchangeably whereas an outlier is rare by definition (see [325, 326]). However, anomalies are by no means rare necessarily. For instance, abnormal high rates of incorrect login attempts to a single account or unusual pressure on a resource as typically intended by DoS or DDoS attacks can also be seen as an anomaly and, in fact, this is already covered by the definition given through [43]. We are compliant with [43] in the sense that an anomaly, whether caused by malicious or legitimate activities, refers to a new significant phenomenon in the data that has not been observed previously. But we also realize that the complex nature of anomalies and their different meanings dependent on the context make the search for a general definition a difficult undertaking. Based on that reason, we leave further debates about their contextual nuances and concentrate on detection mechanisms instead. According to [13, 64], commonly accepted strategies used in ML can be divided into (i) supervised, (ii) unsupervised and (iii) semi-supervised anomaly detection. Strategies of (i) pertain to a conventional supervised learning task with available training data where it is typically assumed that instances of the anomalous class are far more infrequent than those portions of the normal class constituting a class imbalance problem. Techniques of (ii) address classic unsupervised learning where no labeled training data are available and the task is to find data points that are not belonging to any cluster. Methods of (iii) re-

fer to a setup where abundant amounts of data are at hand but labels are only available for the majority class (i.e. the normal or benign class in our case). Thus, one is eager to train a model and all deviations to that resulting model are considered anomalous, i.e. "one-class classification". Contrasting the benefits of these three strategies, we already clarified that supervised learning can provide solid performances demonstrated by experiments (see Section 7.4). An intrinsic problem of supervised techniques, though, is the appropriate handling of data that completely deviates to the training (see Section 2.2.3). This manifests that the spotting of known attacks and their variants is possible but the identification of attacks that have never been seen before (see novel attacks) is a rather difficult endeavor. This observation is also confirmed by other authors (e.g. [77, 78, 327, 328]). Additionally, [327] state empirically that supervised learning is the preferable choice over unsupervised learning when no new attacks are expected and labeled data are available. In situations with new attacks, both supervised and unsupervised detection perform equally moderate in the setup of the authors. The latter point demonstrates the general problem of unsupervised learning not meeting desired hit rates in practice but this result should not hide the fact that unsupervised methods have a clear advantage when it comes to the identification of novel threats, which is also emphasized in [328]. Similar to unsupervised techniques, the amenity of one-class classification is the ability to identify new attacks. Using such methods is still very problematic in situations where live data already are deviating marginally from data seen during training, which causes high false alarm rates. The reason for this issue is the closed-world assumption (see Section 5.3) and the rather limited generalization opportunity as few reference points exist that can guide the learning beyond conservative decision boundaries. Nonetheless, they can be an alternative in several domains because most supervised modeling is prone to class imbalance. This behavior is studied in [329, 330] where the removal of minority classes or the increase of class imbalance clearly destabilize supervised predictors. However, [331] disclosed that the utilization of sampling techniques can indeed extend the superior of binary learners over one-class methods under extreme class imbalance particularly for classification problems with high domain complexity.

Apart from anomalies and their detection aspects, the underlying learning setup also requires ample attention in order to reflect a realistic environment for our HFIDS. This involves important questions in terms of "when" to train and deploy ADC. While answers are straightforward in a conventional setup clearly separating between a train and live phase, answers are not as obvious as they seem when operating on a stream of data, which is our objective (see Section 5.5). In this setting, research predominantly presumes that the entire stream is labeled. Anyhow, general access to class labels can only be guaranteed for a small portion of learning tasks and is a rather optimistic assumption for streaming applications [332, 333]. The reason behind this point is very plausible because it is way beyond a human expert's capability to label all data considering the excessive costs of labeling (see Section 2.2.1) and the speed of data arrival (see Section 2.3.1). This impracticality has stimulated research to find alternatives. Some works provide solutions where only a small but constant fraction of arriving data is labeled (e.g. [334, 335, 336]) whereas others require labels as per request (e.g. [337, 338, 339]), i.e. a variant of active learning. Further research activities are concerned with a delayed arrival of labels, which is termed "verification latency" (e.g. [340, 341]). Most recent

advances are focusing on a stricter version of the latter problem often termed "initially labeled streaming environment" (e.g. [342, 343]) or "extreme verification latency" (EVL) (e.g. [344, 345]). In this setting, the time interval between arriving example and corresponding label reaches infinity eventually, which means that the true class labels of arriving examples are never available after some initialization phase. It can be seen as a straight port from the conventional (semi-)supervised learning setup with the exception that a nonstationary environment is presumed implicitly due to the evolving stream characteristics (see Section 2.2.3). Thus, EVL is a very realistic setting affecting many real-world applications including our domain of interest where the NIDS in charge must be trained appropriately on a given network environment right before its deployment.

Based on this outline, EVL constitutes the groundwork for the design of ADC such that system operators, ideally, should not be bothered after initialization unless an alarm is raised. Yet, building a reliable system in this environment tends to be highly difficult such that this strict setup has to be relaxed to a certain degree which we further elaborate during the course of this chapter. In terms of ADC's detection mechanism, the functionality of unsupervised and one-class classification are very tempting but they should not be overstated. Their high false positive rates are a critical factor with respect to operational aspects and the credibility of the HFIDS in the long run (see Section 5.3). Therefore, we combine supervised and unsupervised anomaly detection techniques in order to keep potential false alarms at an acceptable level and thereby sustaining the capability to spot new attacks at the same time. One-class classification is not considered for the design of ADC as we are convinced by the results of [331] to employ sampling techniques instead. This also correspond with our experimental findings conducted upfront.

8.2 Background

Building up the supervised part for ADC based on a single learning algorithm has several drawbacks. Instead, we propose the utilization of multiple classifiers that can operate in parallel. We, ergo, provide sound arguments for this design and outline the two most prominent approaches in this direction (Section 8.2.1). Given these preliminaries, closely related methods for nonstationary environments are retrospected with a discussion on their practical applicability (Section 8.2.2). Furthermore, we are detailing algorithms that operate exclusively under the EVL assumption and sketch several pitfalls considering their usage (Section 8.2.3).

8.2.1 Ensemble Learning Preliminaries

Developing a model that generalizes well on unseen data is the ultimate goal of supervised learning reviewing Section 2.2.1. However, hypotheses that operate holistically sound in the target domain are often difficult to obtain in practice due to unfavorable circumstances commonly encountered during learning. These situations include but are not limited to insufficient training data, an inducer's sensitivity to class imbalance or its inability to address the inherent complexity of the classification problem. Thus, the

question arises how we can get over this dilemma and, in fact, several good arguments point towards the combination of a number of hypotheses that perform moderately instead of finding a single theory that is strong. Fundamental reasons for this very appealing but rather unorthodox approach are based on (i) statistical, (ii) computational and (iii) representational grounds as originally characterized in [346, 347]. Detailing (i), let us consider a situation where we are given a number of predictors performing equally well, which is often the case when available training and test data are limited. In such a setting, choosing the right model to be deployed is not trivial. Combining all theories by "averaging" their predictions is probably the safest way to maneuver around the crucial selection process [348]. This strategy may or may not beat the best performing hypothesis among them but certainly reduces the risk of an unfortunate selection [349]. With sufficient training and test data, learning can be still a problem referring to (ii). For instance, inducers can get stuck in local optima including most ANNs and DTs [346, 347]. Hence training multiple classifiers with different starting points may lead to a better approximation than any individual model once their results are aggregated appropriately [348]. In a similar context, [349] emphasizes that a vast amount of training data can overwhelm a single learning algorithm from an computational perspective as well. Hence, it is more suitable to repartition the problem space into smaller subsets and train learners on these more manageable data chunks instead. Building upon a sophisticated voting mechanism, produced outputs often prove to be very useful. (iii) Due to the inductive bias of a single learning method, resulting hypotheses might not be even close to the target function. Such circumstances, for instance, can be compensated by a group of learners that are trained with a focus to specific parts of the data yielding experts of the respective region in the problem domain. Obviously, they cannot be expected to perform well on all unseen data points as they are unaware of structures in other regions by purpose. Nonetheless, a reorganization of the decision-making process in such a way that new classifications are directed to those models considered to be more competent for the region in questions virtually can result in solid overall predictive performances. These and similar arguments to increase classification performances with relative simple means have attracted researchers from diverse fields broadly starting in the 1990s. Consequently, developments in this direction are known under various synonyms including "mixture of experts" (e.g. ([350, 351]), "combination of multiple classifiers" (e.g. [352, 353]) or "ensemble learning" (e.g. [347, 354]). Since then, choosing a group of moderate theories over a single classifier has become one of the most active directions in ML with a strong attention to applications in nonstationary environments during the last decade. Thus, a review on ensemble learning literature under concept drift has to follow (see Section 8.2.2). But prior to that, let us first discuss further reasons why ensemble learning actually works and what are its building blocks. Against this backdrop, we briefly outline the two most prominent representative ensemble methods in the remainder of this section.

Earlier, we motivated ensemble learning based on practical reasons and stated why growing a single predictor often fails in practice. To understand the underlying rationale more formally, let us elaborate on "simple majority voting" that is an established strategy to unify the individual results produced by members of an ensemble. If $s \in \mathbb{N}$ is the number of models in our ensemble with $s > 2$ and s is odd, simple majority voting refers to the policy of finally selecting the class label that has at least a $\lfloor s/2 \rfloor + 1$ consen-

sus among the members. In other words, the ensemble only makes a wrong prediction if more than half of its members are mistaken. If sufficient members are well-chosen with the probability $p \in [0, 1]$ of being correct and produced errors are not directly correlated, i.e. theories are independent, then intuitively it is hard to imaging that this mechanism will fail anyway. In fact, we can show this more formally by exploiting the binomial distribution that reflects the ensemble's probability of making the correct prediction (e.g [355, 356]):

$$P_H(s) := \sum_{i=\lfloor \frac{s}{2} \rfloor + 1}^{s} \binom{s}{i} p^i (1-p)^{s-i} \,. \tag{8.1}$$

Hence, the following observations can be derived straightforwardly: (i) $P_H(s)$ is monotonically increasing and $\lim_{s\to\infty} P_H(s) = 1$ for $p > 0.50$, (ii) $P_H(s)$ is monotonically decreasing and $\lim_{s\to\infty} P_H(s) = 0$ for $p < 0.50$ and (iii) $P_H(s) = p$ for $p = 0.50$. These three properties of (8.1) are also known as the "Condorcet jury theorem"[1]. It states that by adding more members, the ensemble performance approaches the optimum if the involved learning algorithm produces models that are at least better than random guessing, i.e. a "weak learner", which is a relative mild assumption in most environments. Obviously, the independence among the learners is a crucial factor for this theorem as induced hypotheses of weak learners have an inherent difficulty to describe the problem domain entirely precise. Consequently, occurring mistakes of one theory must be compensated by others. This in turn can only be realized if members are sufficiently distinct. For this reason, "diversity" among the group can be considered the cornerstone of ensemble learning (e.g. [348, 349]). According to [357], several techniques can be employed to reach diversity:

- **Data sample manipulation:** Applying sampling methods to available training data such as "bootstrapping" [358] let us draw different variations from the same source. Inducing multiple models on these replicates can create varying decision boundaries on the same classification problem.
- **Input feature manipulation:** Available data can also be spitted with respect to different feature subsets that are drawn systematically (see FS) or randomly [359]. This way, different views emerge as features used for one predictor are inaccessible for others. Intuitively, this diversifies the grounds for decision-making.
- **Learning parameter manipulation:** Adjusting a learner's parameter settings for each training indeed can change underlying assumption to build up a theory (e.g. different split strategies on DTs or changing the number of layers for ANNs). As a result, different models can emerge based on the same inducer and data.
- **Output representation manipulation:** These methods generate diversity using different output representations. For instance, [360] uses "error correcting output codes" to turn a multiclass problem into a binary one by introducing "meta classes" that are either mapped to one or multiple original classes. Other examples are "output smearing" or "output flipping" as given in [361] where noise is introduced to perturb growing predictors.

[1] This theorem is named after the French mathematician M. de Condorcet who already recognized these facts in the year 1785.

Beside these four mentioned methods, other ideas or combinations of them are conceivable in order to obtain sufficient diversity. Utilizing different learners with different inductive biases or leveraging sampling in conjunction with varying features are only two possibilities. In summary, we can identify two key components that are required to build reliable ensembles, i.e. (i) the composition of a diverse group of individual models and (ii) selecting an appropriate policy capable to fuse the different outputs towards a unified prediction. In what follows, let us bring together these two components by discussing "bootstrap aggregating" (bagging) [362] and "boosting"[2] (e.g. [363, 364]) as the two most successful methods in the field of ensemble learning [348]. So far, we learned that combining independent predictors may have an extreme effect to reduce error rates. Bagging provides an intuitive approach to pursue exactly this objective by bootstrapping and aggregating. Therefore, its expected input is a set of given training data $T \subseteq \mathcal{V} \times \mathcal{C}$, a given base learner l and let $s \in \mathbb{N}$ be the size of the ensemble with $s > 2$ and s is odd. W.l.o.g., we further assume a binary classification problem with $\mathcal{C} = \{-1, +1\}$. Then, bagging follows a two-step processing in s iterations. In each round t, the procedure collects bootstrapped replicates S_t of T, i.e. a sample randomly drawn with replacement, and trains l on S_t yielding hypothesis $h_t : \mathcal{V} \rightarrow \mathcal{C}, 1 \leq t \leq s$. After all, the ensemble $H = \{h_1, ..., h_s\}$ is obtained and can be combined with simple majority voting

$$sgn(\sum_{t=1}^{s} h_t(x)) \; , \tag{8.2}$$

where the "sign function" $sgn : \mathbb{R} \rightarrow \{-1, 0, +1\}$ extracts the sign of any real number provided as input. Note that also any other sophisticated policy to classify unseen data during the live phase is plausible. Considering the same inputs to accomplish the same goal, the internals of boosting operate quite differently. In essence, it sequentially builds models h_t whereas the selection of training data in round $t+1$ is based on the error produced in t, i.e. the training of h_{t+1} is more focused on data misclassified by h_t. Detailing this more technically, let us consider $D_t : \mathcal{V} \rightarrow [0, 1]$ to be a distribution function that expresses how likely examples are being part of the training data in iteration t with the initial setup $D_1(x) = {}^1/_{|T|}, \forall \langle x, c(x) \rangle \in T$. Boosting then samples S_t from T according to D_t and trains h_t on it. After that, it updates D_{t+1} defined by

$$D_{t+1}(x) = \frac{D_t(x)}{Z_t} \cdot \begin{cases} \frac{\epsilon_t}{1-\epsilon_t} & \text{, if } h_t(x) = c(x) \\ 1 & \text{, otherwise} \end{cases} \tag{8.3}$$

where ϵ_t is the weighted error produced by h_t and Z_t is a normalization constant. This processing continues until the final iteration completes. We receive $H = \{h_1, ..., h_s\}$ to predict unknown instances $x' \in \mathcal{V}$ using a weighted plurality vote

$$\arg\max_{c_j \in \mathcal{C}} \sum_{t:h_t(x')=c_j} log(\frac{1-\epsilon_t}{\epsilon_t}) \; . \tag{8.4}$$

Note, this variant of boosting is suitable to solve both binary and multiclass classification problems. A further depiction of both introduced meta learners is provided through Algorithm 8.1.

[2] Among the variety of versions developed over the years, we concentrate here on AdaBoost.M1 that sufficiently illustrates the general idea behind boosting.

Input: $T \subseteq \mathcal{V} \times \mathcal{C}$ (labeled training data), $s \in \mathbb{N}$ (number of ensemble members), l (weak learner)
Output: $H = \{h_1, ..., h_s\}$ (the ensemble)

BEGIN
  $H \leftarrow \emptyset$
  FOR $t \leftarrow 1, ..., s$ **LOOP**
    pick bootstrapped replicate S_t from T
    create h_t on S_t using l as base algorithm and append h_t to H
  END LOOP
END

Input: $T \subseteq \mathcal{V} \times \mathcal{C}$ (labeled training data), $s \in \mathbb{N}$ (number of ensemble members), l (weak learner)
Output: $H = \{h_1, ..., h_s\}$ (the ensemble)

BEGIN
  $H \leftarrow \emptyset$
  $D_1(x) \leftarrow \frac{1}{|T|}, \forall \langle x, c(x) \rangle \in T$
  FOR $t \leftarrow 1, ..., s$ **LOOP**
    create h_t with l using T according to D_t and append h_t to H
    $\epsilon_t \leftarrow \sum_{x:h_t(x) \neq c(x)} D_t(x)$
    update D_{t+1} according to (8.3)
  END LOOP
END

Algorithm 8.1: The two most prominent ensemble learning algorithms: bagging (left) and boosting illustrated by AdaBoost.M1 (right)

8.2.2 Adaptive Ensembles

A large body of algorithms has emerged from ML research to adapt to nonstationary environments over the years. In general, these can be broadly categorized as "active" and "inactive" methods (e.g. [102, 365]). Sometimes, authors also refer to either "trigger-based" or "evolving" while others term them "explicit" or "implicit" approaches (e.g. [106, 366]), which further complement the meaning of both groups very well. In that sense, the former group of methods assumes stable processes that may transition to another stable processes over time whereas the phase in between is the actual drift. The beauty of this setting is that involved models are not required to undergo any modification unless a change takes place. Yet, the occurrence of a change point has to be detected. Typically, this is achieved by another mechanism triggering the adaption process, i.e. a so-called "drift detector" (e.g. [367, 368, 369]). Besides this supplemental component that has to be developed, a further hurdle to these explicit methods are slowly evolving drifts (see [368]), which are not meeting the conjecture of a single change point necessarily. The latter group is not based on such an assumption. Instead, it tries to adapt continuously to changes in the underlying data distribution that can be understood as a type of "forgetting mechanism". Ensemble learning is very interesting for either one of the two techniques even though most research on ensemble learning coping with concept drift can be attributed to the second group. The main reason behind these developments is certainly their inherent flexible structure that can be exploited in various forms in order to adapt to changing circumstances quite naturally. According to the work in [370], the adaptability of an ensemble can be organized along the following three points:

(i) Changing the combination policy
(ii) Updating members continuously
(iii) Adjusting the ensemble structure

Out of simplicity, we call any ensemble pursuing one or more of these three strategies an "adaptive ensemble". While (i) deals with behavioral adjustments of the ensemble explicitly by keeping all members but changing their influence on the final decision instead (e.g. [371, 372, 373]), (ii) is concerned with the recency of models by steadily

refreshing their concept approximation through continuous training (e.g. [374, 375]). With respect to (iii), adaptivity is reached by adding or replacing predictors that change the structure of an ensemble. As one of the very first adaptive ensembles, the SEA algorithm [376] follows this technique. It uses an ensemble of C4.5 DTs with a fixed size that creates new trees each time a new batch of data arrives and replaces poor performing members in the ensemble with a newer more accurate version. Other adaptive ensembles combine more than one strategy such as AWE [377], DWM [378, 379], Learn++.NSE [365] and AUE2 [380] which we outline briefly: AWE is similar to SEA as it manages also an ensemble of constant size and only includes new classifiers if its performance is among the top k members. Predictions are made according to a weighted majority vote whereas a member's weight is measured by the contribution the member provides compared to that of a classifier performing by chance using "mean square errors". DWM in turn is dynamic in its size but uses a weighted combination policy as well that is based on the weighted majority algorithm introduced in [372]. It adds new experts if the global performance reduces and adjusts the weights of members with respect to their local performances. They are finally removed from the ensemble if their weights fall below a specific threshold. Learn++.NSE is an extension of Learn++ [381] for nonstationary environments and has some similarities to DWM and boosting (see Algorithm 8.1). According to the authors, the main advantages are its adaptivity to a broad range of drifts disregarding whether they are slow, fast, reoccurring or new concepts evolve. Another contribution is that current and older classifiers are combined using a weighted majority vote based on "time-adjusted errors". This way, actual knowledge is used to predict arriving instances whereas older knowledge that is not relevant at the moment has no or very few influence on the decision-making. Extensions to Learn++.NSE addressing class imbalance can be found in [382]. As a revised version of the original AUE approach [383], AUE2 uses a fixed ensemble and substitutes the least accurate member with a new learned theory. In addition to this, each remaining classifier is also incrementally updated, which constitutes the novelty of this approach. This way, AUE2 is capable to react to a broad range of different drift scenarios. Its voting policy is inspired by AWE. Apart from these representative achievements in the field of adaptive ensembles, further effort on ensemble learning coping with concept drift exists. Offering further insights, we refer to recent surveys in [384, 385].

Even though reported advances of adaptive ensembles reveal that a broad range of drift scenarios can be covered, most approaches are academic though and have a very limited applicability in practice[3]. This is due to the fact that they assume labeled data to arrive continuously along with the data instances to classify. Considering data stream applications or other scenarios where raw data are abundant, one can easily verify that such an assumption is prohibitive given the high costs for labeling. A rather realistic scenario is provided by EVL instead where access to labeled data is denied after some point in time. We get back to this subject in the next Section 8.2.3 by outlining recent efforts on EVL and see that only few algorithms have covered this complex learning setup to date.

[3] It is noteworthy at this point that this statement does not hold for the listed ensemble methods alone. In fact, the majority of existing ML algorithms covering nonstationary environments share this characteristic to this day.

8.2.3 Initially Labeled Environments

Early works such as [386, 387] are able to track specific drifts via a mixture of subpopulations and the EM algorithm, which are exploited systematically to label arriving unlabeled data. These works are extended in [388] as they are limited to continuous data and local drift. Independently, a similar approach is developed by [389] using ANNs. Unfortunately, these four statistical approaches neither are capable to detect sudden drifts nor newly evolving concepts. The COMPOSE framework [342, 343] also copes with EVL by using a geometry-based approach. First, available labeled data are leveraged to perform a propagation to unlabeled data. In the second step, "α-shapes" are created around clusters with same label, which are tightened to extract the "core support region" of each cluster. Instances in that region are utilized to label data for upcoming iterations. This learning strategy of COMPOSE is further improved in [345] as the computation of α-shapes is very demanding especially with the increase of dimensionality. Thus, the authors remove the core support extraction procedure and reuse all labeled instances as core instead. Results show that this simplification surprisingly pays off for several benchmark scenarios compared to the original method from both an execution time and *ACC* perspective. Note that, COMPOSE as well as its improvement rely on the "limited gradual drift assumption" [390]. Another approach is SCARGC [344]. The algorithm can be summarized in several steps. Initially, a supervised model is built based on existing labeled data and k-Means is employed to cluster the data whereas each cluster is associated with its true class. In the live phase, i.e. a stream of unlabeled data arrives, SCARGC classifies each instance at a time and pools them to a buffer. The cluster landscape is updated whenever the buffer size exceeds and at least a minimum amount of new instances are notable per class. Yet, these clusters require adequate labels, which is tackled by a mechanism that tracks the movements of new clusters compared to old ones. Given a new unlabeled cluster, the algorithm seeks for the closest cluster obtained in the previous round according to the minimum Euclidean distance of their centroids. Using this association, unlabeled clusters are labeled straightaway. In the final stage, this new labeled cluster landscape is then exploited to update the initial model and the buffer is cleared. Afterwards, SCARGC reverts to the first step in the live phase to continuously process the data stream. Reported results show that SCARGC outperforms [386] and that it is very similar to COMPOSE in terms of *ACC* on selected benchmark data. However, experiments in [345] indicate that it is still behind the improved version of COMPOSE. MClassification [391] is another method to handle EVL and proposed by the same authors that introduced SCARGC. It is solely based on the concept of "micro-clustering" (see Section 8.3.2) and requires only one user-defined parameter, i.e. a radius threshold. In essence, it associates each labeled instance to a micro-cluster initially. On each arriving example in the live phase, it uses a similar strategy as SCARGC to identify the closest micro-cluster assigned and outputs its label. The new instance is added to the micro-cluster, which is later checked whether its maximum radius is surpassed. In such cases, the instance becomes a new micro-cluster and the routine returns to process new unlabeled data. This rather simple approach showcases comparable results to SCARGC with the benefit of fewer parameters. Note, SCARGC and MClassification are explicitly designed for specific drift scenarios.

Earlier, we declared our concerns that most adaptive ensembles are impractical as they suppose GT data to be accessible predominantly. Therefore, further methods started to address this issue by proposing solutions under EVL. Heavily making use of semi-supervised learning techniques, research in this direction can be still considered at an early stage despite the reported progress. All outlined approaches pursue the limited gradual drift assumption, which corresponds to an incremental drift in our terminology (Section 2.2.3). This prerequisite certainly facilitate the reconciliation between recent models and newly arriving data but neglect other drift types. As a result, their applicability in gradual or sudden drift scenarios is not directly viable in practice. Projecting these circumstances to our objectives turn these findings into crucial ones particularly when reviewing our defined threat model where these drifts are manifested (see Section 5.2). Therefore, we address this challenge in Section 8.3 by developing ADC combining existing techniques. Note, results on this attempt have not been published elsewhere.

8.3 Detection Component

In this section, we introduce the basic idea of ADC, which can be divided into two parts. The first part is concerned with the development of an ensemble that provides solid performance under EVL and a certain drift scenarios (Section 8.3.1). To further extend its capabilities, we enclose an unsupervised technique to this detector permitting to cover further drift scenarios in addition (Section 8.3.2). Ultimately, we contextualize ADC with existing approaches and state potential pitfalls (Section 8.3.3).

8.3.1 Ensemble Composition

In Section 8.2.1 and 8.2.2, we illuminated the main ideas behind the ensemble learning framework and outlined the advances of adaptive ensembles as broad branch of research countering concept drifting scenarios among others. Within this framework, their usage can be extremely valuable for intrusion detection as well. For instance, models can be collected over time and adequately combined to obtain adaptivity [214]. Moreover specific features according to their data origins can be utilized to build ensembles that are more accurate than any individual member or learners trained on the entire feature set [392]. Another example are the employment of meaningful features through FS to train each ensemble member with a focus on specific attacks [393]. Hence, we leverage this very promising framework and present an ensemble-based algorithm as a part of the developments of ADC in this section that is inspired by existing works. Before we can start detailing this approach, we have yet to clarify several issues to build up its foundation. These can be conflated along the following three points: (i) the ensemble method, (ii) the aspect of diversity and (iii) the class imbalance problem. Considering issue (i), we already highlighted bagging and boosting as the two most prominent ensemble methods. While both demonstrated their practical effectiveness in a number of disciplines, we also have to take into account their efficiency, which is a fundamental design principle for the processing of the HFIDS and ADC in particular. Retrospecting their structures yields that bagging is a straight candidate for this duty. The steps required to build

LC	F00 & FH0	F0W & FHW
TREE	C4.5	C4.5
RULE	PART	PART
ANN	RBFN	MLP
SVM	SVM-R	SVM-R
MISC	NBay	NBay

TABLE 8.1: Heterogeneous ensemble composition for ADC as a result of the flow feature analysis using the best learning algorithms per LC and FFC

each member are independent constituting an ideal premise for its deployment in parallel environments. On the contrary, the processing of boosting is sequential in nature where each iteration builds upon the previous one in order to reinitialize its inherent distribution function. Based on that, we choose bagging over booting to build ADC's ensemble. Generally, we are not denying that boosting can also be enabled for parallel execution scenarios. However, it is the simplicity of bagging that appeals with respect to our application. These findings are also in line with [394] confirming that boosting is rather hard to implement for the real-time demands of IDSs. That given, let us move to issue (ii) where diversity among the members need to be ensured to turn an ensemble into an effective predictor. Therefore, we try to reach diversity on three different levels. The first level is realized by bootstrapping, which is naturally furnished by bagging. The second and third level rest upon results already obtained in Chapter 7 where we figured out meaningful features for different FFCs (see Table 7.4) and learning algorithms that operate reliably exploiting them (see Table 7.6 and Figure 7.4). More precisely, we select the best performing inducer per LC yielding five ensemble members for each FFC, which are outlined in Table 8.1. This choice requires some explanation because C4.5 is behind ID3 on F0W and all considered SVMs are very similar rank-wise according to our previous experiments. However, these results only emerge due to their varying runtimes. From our practical experience, these can be neglected for category TREE as C4.5 is not far off from ID3 while for the SVM category a decision is not that obvious. Yet, it is the sheer predictive performance of SVM-R paired with the techniques of the next point (iii) that let us accept its higher runtimes because training sets can be considered small. Instantly, this leads to aspect (iii), which is a serious issue not only for network intrusion detection. A straightforward method to circumvent skewed class distributions on binary or multiclass classification tasks is to either "oversample" the minority classes or to "undersample" the majority classes in the training set in order to obtain a balance between the classes (e.g [395, 396, 397]). Nevertheless, both may enclose shortcomings, namely, overfitting for oversampling and the elimination of essential information by data undersampling. To compensate both extremes and inspired by the auspicious results of [398], we pursue a "hybrid sampling method" (HSM). This method equalizes the imbalance by utilizing both sampling techniques simultaneously but attempting to uses oversampling cautiously such that all attacks ever seen can be leveraged to create bootstrapped replicates. An overview of our employed HSM is sketched in Algorithm 8.2 where a binary classification setting is envisioned out of simplicity.

Having clarified these important points, let us concentrate on the ensemble construction of ADC. The main idea relies on an adaptive ensemble consisting of a classifier committee with fixed size where only labeled data are available initially and potential adjustments

Input: $S \subseteq \mathcal{V} \times \mathcal{C} \times [0,1]$ (input data)
Output: V

BEGIN
  $C_1 \leftarrow \{\langle x, c, \psi\rangle \in S \mid c = \text{“malicious”}\}$
  $C_2 \leftarrow \{\langle x, c, \psi\rangle \in S \mid c = \text{“benign”}\}$
  IF $|C_1| \leq \lfloor \frac{|C_2|}{2} \rfloor$ **THEN**
    $S_1 \leftarrow$ randomly upsample $\lfloor \frac{|C_2|}{2} \rfloor$ tuples from C_1
    $S_2 \leftarrow$ randomly downsample $\lfloor \frac{|C_2|}{2} \rfloor$ tuples from C_2
  ELSIF $\lfloor \frac{|C_2|}{2} \rfloor < |C_1| < |C_2|$ **THEN**
    $S_1 \leftarrow C_1$
    $S_2 \leftarrow$ randomly downsample $|C_1|$ tuples from C_2
  ELSIF $|C_1| = |C_2|$ **THEN**
    $S_1 \leftarrow C_1$
    $S_2 \leftarrow C_2$
  ELSE
    $S_1 \leftarrow C_1$
    $S_2 \leftarrow$ randomly upsample $|C_1|$ tuples from C_2
  END IF
  $V \leftarrow S_1 \uplus S_2$
END

Algorithm 8.2: Procedure HSM to obtain uniformed samples from input data with a skewed class distribution

to the model are carried out via self-produced labels on latest data. Given this EVL setup, its procedure is divided into two subroutines outlined in Algorithm 8.3. The first routine is concerned with the training phase where examples with appropriate GT are present. Consequently, the ensemble can be built up from scratch quite similar to the classic bagging approach. Initially, available training data are stored in a data buffer $\mathcal{D}$ and then used to grow the ensemble $H = \{h_1, ..., h_s\}$ based on $s = 5$ base inducers $L = \{l_1, ..., l_s\}$ (see Table 8.1) and HSM($\mathcal{D}$) in such a way that $h_t \in H, 1 \leq t \leq s$ relies on $l_t \in L$ using a randomly assigned reduct (see Table 7.4) to create diversity. In this respect, we introduce $\nu_t(x)$, which transforms incoming examples $x \in \mathcal{V}$ to the desired input structure of h_t recalling that available reducts are defined on a reduced feature set. Hence, ν_t can be understood as projection operation to the respective input subspace induced by the reduct in charge. Lastly, each member $h_t \in H$ is tested against all data in $\mathcal{D}$ to obtain weights $w_t \in [0,1]$ reflecting h_t's intermediate classification performance. The second routine is handling the live phase and expecting unlabeled data only. In essence, this method undertakes two essential tasks, i.e. classification of unseen data and the execution of potential adjustments to the ensemble. Depicting the former task first, predictions rely on a weighted plurality vote $\mathcal{H} : \mathcal{V} \rightarrow \mathcal{C} \times [0,1]$ such that

$$\mathcal{H}(x) = \langle c, \psi \rangle \tag{8.5}$$

with the predicted class label

$$c = \arg\max_{c_j \in \mathcal{C}} \sum_{t=1}^{s} \chi(t,j) \cdot w_t \tag{8.6}$$

and H's confidence on that prediction

$$\psi = \frac{1}{s}(\max_{c_j \in \mathcal{C}} \sum_{t=1}^{s} \chi(t,j) \cdot w_t) \tag{8.7}$$

where

$$\chi(t,j) = \begin{cases} 1 & \text{, if } h_t(\nu_t(x)) = c_j \\ 0 & \text{, otherwise} \end{cases} \tag{8.8}$$

Predefined settings: $H \leftarrow \emptyset$ (ensemble), $\mathcal{D} \leftarrow \emptyset$ (data buffer), $L = \{l_1, ..., l_s\}, s \in \mathbb{N}$ (base learners), $\alpha_\perp \in \mathbb{N}$ (initial sizing factor of $\mathcal{D}$), $w_{min} \in [0, 1]$ (minimum member weight), $\psi_{min} \in [0, 1]$ (minimum label confidence)

Input: $T \subseteq \mathcal{V} \times \mathcal{C}$ (labeled training data)

1: **BEGIN**
2: create tuples $\langle x, c, \psi \rangle, \forall \langle x, c \rangle \in T$ with $\psi = 1$ and append tuples to $\mathcal{D}$
3: construct $H = \{h_1, ..., h_s\}$ bootstrapping from HSM($\mathcal{D}$) and using different feature subsets such that l_t is the learning algorithm for model $h_t, \forall t \in \{1, ..., s\}$
4: associate weight w_t with predictive performance of h_t on HSM($\mathcal{D}$), $\forall t \in \{1, ..., s\}$
5: **END**

Input: $T \subseteq \mathcal{V}$ (unlabeled data)
Output: $V \subseteq T \times \mathcal{C} \times [0, 1]$

1: **BEGIN**
2: compute $\mathcal{H}(x) = \langle c, \psi \rangle, \forall x \in T$ and append tuples $\langle x, c, \psi \rangle$ to output V
3: extract $S = \{\langle x, c, \psi \rangle \in V \mid \psi \geq \psi_{min}\}$ and append S to $\mathcal{D}$
4: recurrently evaluate and update H
5: perform potential cleanups on $\mathcal{D}$
6: **END**

Algorithm 8.3: ADC's adaptive ensemble component: initial training with labeled data (left) and classification and adaption in live phase (right)

1: **BEGIN**
2: split $\mathcal{D}$ to training set S_{train} and test set S_{test}
3: **FOR** $t \leftarrow 1, ..., s$ **LOOP**
4: recompute w_t according to predictive performance of h_t on S_{test}
5: **IF** $w_t < w_{min}$ **THEN**
6: bootstrap S_t from HSM(S_{train})
7: retrain h_t based on l_t using S_t and update w_t based on S_{test}
8: **END IF**
9: **END LOOP**
10: **END**

1: **BEGIN**
2: $C_1 \leftarrow \{\langle x, c, \psi \rangle \in \mathcal{D} \mid c =$ "malicious"$\}$
3: $C_2 \leftarrow \{\langle x, c, \psi \rangle \in \mathcal{D} \mid c =$ "benign"$\}$
4: **IF** $|C_1| > |C_2|$ **THEN**
5: $\alpha_\perp \leftarrow \alpha_\perp + 1$
6: **END IF**
7: **IF** $|C_2| > \alpha_\perp \cdot |T|$ **THEN**
8: $M \leftarrow \{\langle x, c, \psi \rangle \in C_2 \mid \langle x, c, \psi \rangle$ is among the $|C_2| - \alpha_\perp \cdot |T|$ oldest tuples$\}$
9: $\mathcal{D} \leftarrow \mathcal{D} \setminus M$
10: **END IF**
11: **END**

Algorithm 8.4: Subroutines of ADC's adaptive ensemble component: retaining procedure (left) and cleanup of data buffer (right)

is the "characteristic function". Intuitively, $\mathcal{H}$ turns into a conventional majority vote if we consider $|\mathcal{C}| = 2$ and $w_t \approx 1$ such that at least $\lfloor s/2+1 \rfloor$ members agree on the decision. Certainly, the classifications made by the prediction function $\mathcal{H}$ can be considered solid if we are close to the start of the live phase. However, the chance for making incorrect predictions increases as we move away from this comfort zone due to plausible concept drifts. To circumvent such situations, performed classifications are not only returned as output but are also maintained in $\mathcal{D}$ to serve potentials modifications of the ensemble as long as the prediction confidence ψ is equal or beyond the minimum label confidence ψ_{min}. Thus, $\mathcal{D}$ can be used as current training data repository to retrain members in case their performance level drops below the minimum w_{min}, which constitutes the second task. Lastly, benign flows contained in $\mathcal{D}$ are pruned in order to keep only most recent legitimate traffic. Note, we exclude malicious flows from this cleanup activity by purpose. A summary of both adaption and cleanup mechanism is provided in Algorithm 8.4.

Using this approach is very similar to some existing adaptive ensembles with the exception that our method is not assuming that GT data are always available. This way, the presented ensemble is compliant with EVL and works surprising solid in practice. To

demonstrate its applicability, let us consider a scenario where an invader tries to evade detection by modifying the footprint of its attack resulting in an incremental drift. Such a conceivable setting is provided by an illustrative example given through Figure 8.1 that is based on data from the NDSec-1 data set. Employing a naive ensemble with static decision boundaries, the impact of this scenario would be rather catastrophic because the attacker can disguise malicious activities successfully by crossing the believed decision boundary that in turn would result in a high number of *FN*. Tricking available detection mechanisms using our proposed adaptive ensemble is not that simple. This can be justified according to the nature of the drift motion. As soon as sufficient attack instances move towards the global decision boundary, the local boundary of some members becomes imprecise and so their individual confidence drops compared to the overall consensus. This does not intercept attack flows to cross the decision boundary. Nonetheless, as long as we can prevent $\psi \geq \psi_{min}$ and $c_j \neq c(x)$ for any new classification $\mathcal{H}(x) = \langle c_j, \psi \rangle$ (i.e. the committee is convinced about a decision albeit essentially wrong), adaptivity remains intact and the proposed ensemble is capable to keep track with the underlying drift situation.

8.3.2 Unsupervised Supplements

During the ensemble composition for ADC outlined in the last Section 8.3.1, we already demonstrated that its adaption mechanism can work quite well. However, there are essential flaws utilizing this approach. These can be unveiled straightforwardly as we increase the motion of the drift reconsidering the attack evasion scenario outlined in Figure 8.1. In this case the elasticity of the ensemble's decision boundary cannot keep up with the drift resulting in pure attack misses rather quickly. Thus, we can conclude that the effectiveness of the adaptive ensemble holds for slow incremental drifts only. Perhaps more critical in practice are settings where sudden drifts and more general gradual changes emerge that can flip over currently believed decision boundaries immediately. Such situations cannot be handled by the proposed ensemble even though a high degree of diversity and optimal classifiers are inferred. This is due to the level of difficulty resolving such drift characteristics under EVL as we have no indicator at our disposal that can guide the adaption process. As a result, all models of the ensemble may become outdated for certain regions in the instance space all of a sudden. To picture the problematic nature of this task, Figure 8.2 outlines both a sudden drift of attacks and a gradual one of normal flows that would increase the error rate of the ensemble tremendously in a short period of time.

Extending our previous progress on ADC with adequate functionality to cope with sudden and gradual drifts in addition, we take into account a supplemental component that is unsupervised. In particular, we leverage the concept of micro-clustering that broadly won fame in the early 2000s introduced in [399]. It emerged based on the motivation that most traditional clustering algorithms are incapable to run in stream mining scenarios as they turn out to be inefficient or cannot deal with the evolving nature of the data. In this respect, micro-clustering divides the clustering process into two phases. In the first one, i.e. "online clustering", arriving data points are turned into micro-clusters. They can be understood as abstract representations of data points

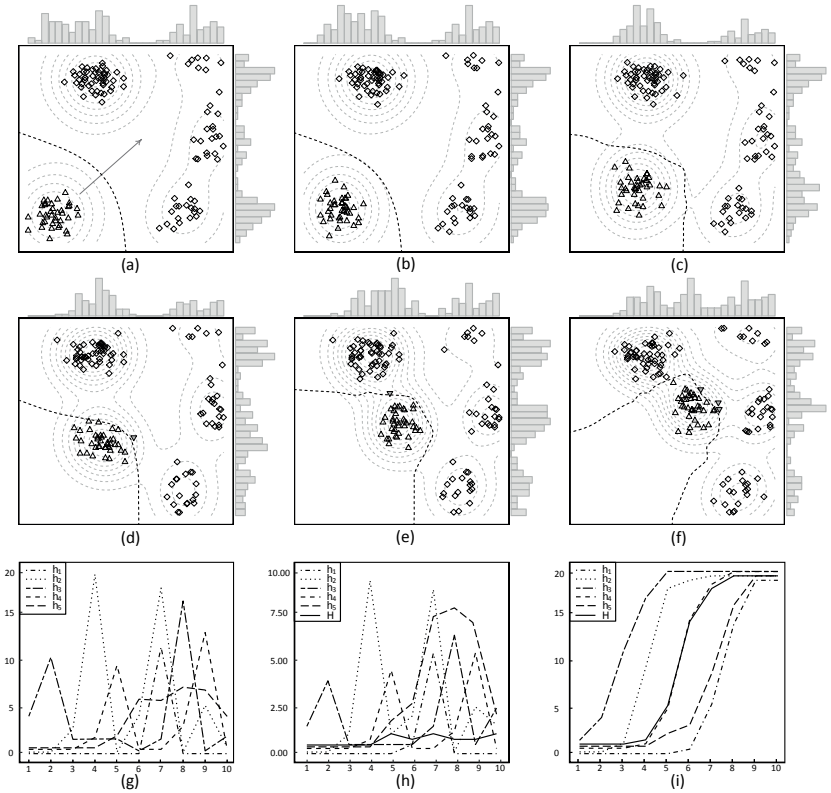

FIGURE 8.1: Evasion scenario through an incremental drift depicted in a two-dimensional instance space with decision boundary (black dashed line) separating attack (△) and benign data (◇): (a) initial trained ADC's ensemble (five members) and (b) to (f) live phase (iteration 2, 4, 6, 8 and 10) where the ensemble progressively adjusts through reweighting and retaining members as new data becomes available leveraging previous predictions; misjudgments of the ensemble are indicated by the ▽-symbol (*FN*); gray bars and gray dashed lines indicate data frequencies and densities; (g) to (h) highlight the estimated and actual error rate of the ensemble while (i) emphasizes its performance without adaption; error rates are in percent

using statistical summaries with the potentials to be extended or merged without access to the original data. This structure can be attributed to [400] and arguably requires significant fewer storage than persisting the entire data collection. In the second phase, i.e. "offline clustering", these condensed summaries gathered over time are utilized to build the final cluster landscape, i.e. "macro-clustering". Apart from [399] introducing the CluStream algorithm that employs a variant of k-Means in the offline phase, several other works followed the paradigm of micro-clusters alone or the two-phase scheme. For instance, DenStream [401] introduces an approach to better separate real micro-clusters from noisy ones and uses a version of DBSCAN [402] for the offline clustering. D-Stream

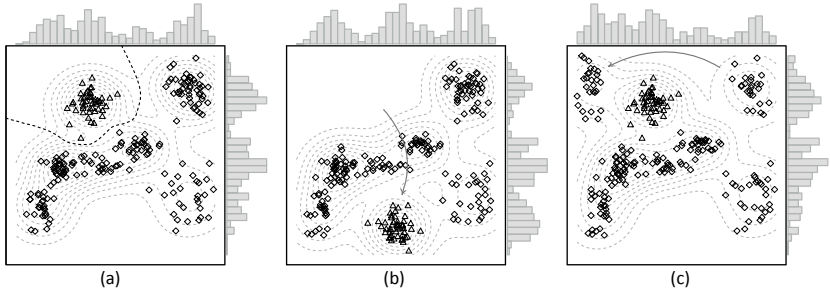

FIGURE 8.2: Drift situation depicted in a two-dimensional instance space that cannot be anticipated by ADC's ensemble alone: (a) initial data and intuitive decision boundary (black dashed line) separating attack (△) and benign data (◇), (b) sudden change of the attack frontier and (c) gradual drift of benign data; gray bars and gray dashed lines indicate data frequencies and densities respectively

[403] uses an alternative techniques employing grids that partition the data space to get over of some problems inherent to CluStream. One of the latest advances from stream clustering research is DBStream [404] that explicitly captures density information between micro-clusters that improves reclustering activities for adjacent clusters. Results in [404] show that DBStream can compete with CluStream, DenStream and D-Stream and it has a higher cluster purity in most instances. Moreover, it demands significantly fewer micro-clusters to achieve the same results as its contenders. We also performed several experiments on these approaches, which confirmed the mentioned findings for the most part. Additionally, DBStream requires few parameters to control the learning process, which is another advantage. Based on that, we pick DBStream as unsupervised supplement for ADC. It should be mentioned that we choose micro-clustering over any other unsupervised approach because it is designed to operate in dynamic scenarios where new clusters evolve or old ones disappear [68]. In particular, the former characteristic is the most essentially one to counter the mentioned drift facets which we illuminate in what follows.

To exploit the advantages of both ADC parts, i.e. the ensemble and the micro-clustering, we propose to run them in parallel. This way, the ensemble can classify newly arriving flows and adapt to minor changes of the data generating process whereas the main task of the unsupervised algorithm is to identify newly evolving clusters. In case of such a detection, ADC essentially can never be sure to which class the novel cluster actually belongs. On that account, we have to introduce an additional revision component to the processing by sending relevant information of that cluster to AM on request in order to label the cluster using manual inspection. Consequently, a new temporal class label has to be established which we call "anomalous". This meta class covers ADC's uncertainty until it is resolved by operational staff. In the meantime, all related flows maintained in $\mathcal{D}$ have to be relabeled and the ensemble is instructed to retrain for the purpose of adapting to this new situation. On the one hand, we are fully aware that this supplementary step is a critical design choice for the HFIDS as human intervention is needed. On the other hand, we are not aware of any automated and workable solution to

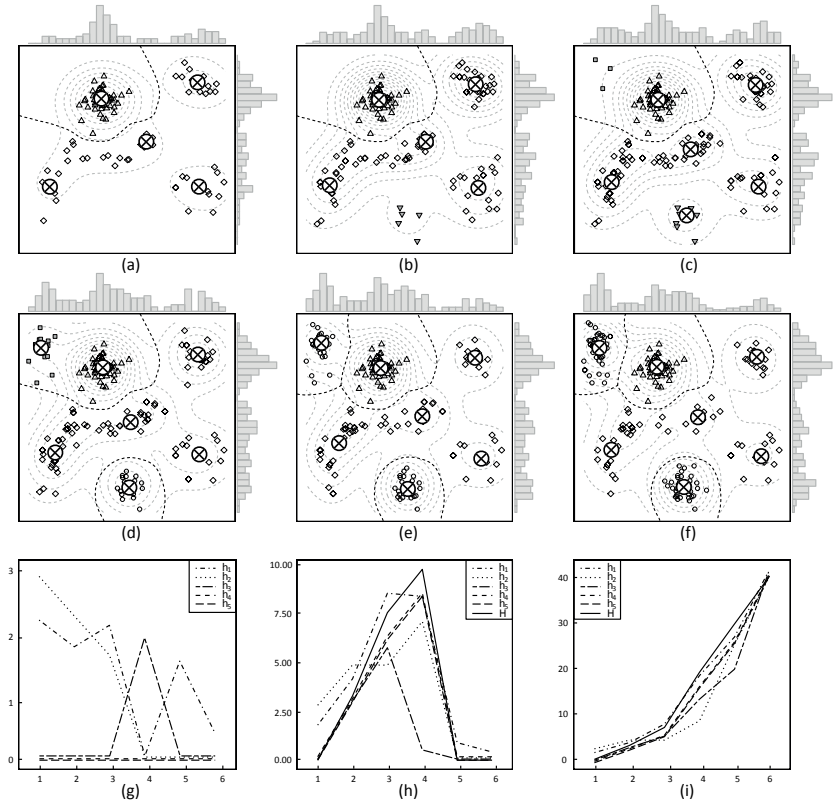

FIGURE 8.3: Resolving sudden and gradual drifts combining supervised and unsupervised learning illustrated in a two-dimensional instance space with decision boundary (black dashed line) separating attack (△) and benign data (◇) where cluster centers are outlined by the ⊗-symbol: (a) initial trained ADC and (b) to (f) live phase with the evolution of two new clusters that are recognized in (c) and (d) instructing to retrain the ensemble with the anomalous class (○); misjudgments are depicted by the ▽-symbol (*FN*) and the ■-symbol (*FP*); gray bars and gray dashed lines indicate data frequencies and densities; (g) to (h) highlight the estimated and actual error rate of ADC while (i) emphasizes its performance without clustering; error rates are expressed in percent

maneuver around such scenarios when considering EVL in nonstationary environments and no additional information is available or no further assumptions are made upfront. That being said, let us move to a practical example given through Figure 8.3 where sudden and gradual drifts occur and both parts of ADC are incorporated. Initially, the data generating process is stable and so arriving flow data can be separated by the decision boundary of the well-trained ensemble. This is also compliant with the clusters identified by DBStream so far. In the next time frame, attack flows suddenly appear in a very sparse region that are part of an abrupt drift. At this stage, the base problem of

ADC becomes apparent when considering the ensemble alone because all of its members are convinced to classify these newly evolving data points as legitimated flows what is essentially a wrong judgment. Over time, more of these flows arise such that the unsupervised part identifies a new cluster. As a result, all involved flows are labeled as anomalous and the ensemble is retrained. In subsequent steps, a new legitimate cluster shapes up in a region believed to be malicious. Consequently also these are misclassified by raised false alarms until the cluster is identified and appropriate actions can be initiated. Evaluating this scenario quantitatively produces a high number of errors as a result of the latency between new data points that arrive occasional and the evolution of a cluster that arises from these points. Most importantly, however, ADC can self-adjust to these situations by decreasing the error rate using its meta class in subsequent steps. Without this auxiliary construct and the support of micro-clustering, ADC's error rate would increase to an unacceptable high level yielding a useless anomaly detector ultimately. Recall that all flows of the minority class are kept in $\mathcal{D}$, which is the reason why original attack flows remain in our illustrative example despite the sudden drift scenario.

8.3.3 Discussion

In this Section 8.3, we proposed ADC based on supervised and unsupervised learning techniques with several similarities to existing literature. The supervised part is similar to the early SEA algorithm in the sense that we also utilize an fixed-size ensemble and batch-based processing. Both approaches replace individual members in case their predictive performance degrades but ADC is not limited to data from the current or last batch due to the data buffer $\mathcal{D}$ that can be sized appropriately. Similarities to AWE are the constant-sized ensemble and the weights per member influencing final decisions as part of a weighted voting schema. The idea of maintaining weights that are proportional to the performance is also shared with Learn++.NSE. Perhaps the closes approach to our development is DWM. Yet, it uses an incremental ensemble with dynamic size and relies, similar to other methods, on GT data throughout the entire learning process, which is rather unrealistic (see Section 8.1). We try to get over this dilemma by leveraging predicted labels that are most confident in order to keep our ensemble up to date. This processing is related to self-training and co-training but under aggravated circumstances as the underlying data distribution is expected to drift. This makes it necessary to introduce further assumptions, which refers to the availability of sufficient diversity and a solid predictive performance among the ensemble members. As such, obsolete local expertise can be identified and substituted while the global performance on unseen data still remains intact. During the course of developing ADC, we experienced that this can only be guaranteed for slow incremental drifts because abrupt and gradual drift characteristics are more radical in nature. Based on these findings, the unsupervised part of ADC was introduced. It takes care of the evolution of anomalous clusters and, thus, can be seen as a detector for newly evolving clusters with the ability to point out involved regions. This functionality is highly related to the subject of "novelty detection" (e.g. [405, 406, 407]). Yet, unlike novelties that are, according to [13], typically associated to the normal class, we cannot make such an assumption as new formed clusters of flows may be either legitimate or malicious. Given that point, we

argue that it is extremely difficult, if not impossible, to determine associated class labels of these new clusters in an automated fashion taking EVL and nonstationarity into consideration. To counter this problem, the introduction of an auxiliary class label is required. It is exploited intermediately until the real class label of the cluster is disclosed by administrative personnel, which can be seen as a variant of active learning. Contrasting ADC with existing methods that are not assuming general GT availability, early works already use micro-clustering but suppose that a small but constant amount of labels is always provided, which certainly can help to identify the class of new evolving clusters. Other methods obtain labels as per request, which corresponds to the active learning component of ADC. The problem with this setting is that labels cannot be expected to arrive immediately due to human intervention. Further restrictions lead to solutions under EVL. Among existing methods such as COMPOSE or MClassification, only incremental drifts can be handled. Thus, ADC can be seen as an extension to these developments. In addition to that, we also want to discuss some practical aspects regarding the micro-clustering of DBStream. This algorithm is density-based and is not requiring a parameter dictating the number of clusters to extract. Yet, its essential parameter is the micro-cluster radius to determine whether a new data point falls into an existing cluster or not. This can cause practical pitfalls. Assuming a landscape with different density regions, it might be difficult to create accurate clusters because the selected radius is either too large or too small. This circumstance also constitutes another issue. In case a low value is chosen for the radius, more sparse clusters cannot be spotted as part of abrupt or gradual drifts, which potentially could be exploited by an attacker to evade detection. However, such a situation would infer that the attacker is aware of ADC's internal parameter settings. This is very unlikely unless parts of the HFIDS architecture are already compromised.

8.4 Summary

This chapter was devoted to the development of ADC, an anomaly detection method, designed to examine unseen FVs that arrive from the network infrastructure of interest. Concerning this matter, the main problem is not only related to the identification of attacks by separating malicious and benign activities using flow data but to changing conditions as well. It is further fueled by the fact that training data from the network environment are only available during an initialization phase right before the HFIDS is deployed. This composition of realistic assumptions is not pursued by most existing ML approaches and even less reflected in flow-based detection or intrusion detection literature in general. Based on that, it is not trivial to build a single model fitting all of these challenges at the same time. Yet, the combination of multiple models with moderate performances has proven to be effective because their overall classification abilities can be boosted under relative mild assumptions. Such ensembles of individual theories have additional benefits particularly in nonstationary environments, which finally made this approach a solid fit for our purpose. Another argument was the evaluation of single learners in the previous chapter yielding very promising and reasonable learners for the ensemble. We demonstrated that this supervised component of ADC performs and adapts quite well on an adversarial scenario. In this setup with initial labels only,

attack evasion tactics were employed constituting an incremental drift. In contrast to various single classifiers or a static ensemble, our adaptive ensemble produced a relative low error rate. however, it could not cope with abrupt or more gradual changes due to missing GT data in the live phase that typically guide the adaption of other ensembles. Relying on this reason and to be compliant with our defined threat model, ADC was extended with an unsupervised component based on micro-clustering. It acts as detector for newly evolving clusters that can be part of a new attack or legitimate changes of the underlying network landscape. Results show that this enhancement to ADC certainly payed off because it outperformed its contained adaptive ensemble and any of its individual members by a big margin.

Closing this chapter, we want to highlight three objectives that are still open to this point. During the design of ADC, our focus was on developing an accurate anomaly detector for nonstationary environments. This undertaking brought a high level of complexity, though, reviewing its internal structure. This certainly jeopardizes transparency aspects of the HFIDS. Hence, we introduce the basic notion of PBC in the next Chapter 9 in order to reconcile a balance between accurate and interpretable decision-making. A further issue is the technical integration of ADC with respect to the overall system architecture including the customization of the initial training phase requiring further details. Closely related to this point are the performed tests in this chapter. Despite their validity, they had a rather illustrative character raising the demand to study the trustworthiness of ADC in more depth. Both last mentioned problems are addressed in Chapter 10 consequently as part of the overall system evaluation.

Chapter 9

Pattern Building

Transparency is a key aspect of the proposed HFIDS permitting to get insights to the decision-making process and to immediately turn raised alerts into purposive counteractions. Therefore, the extraction of regularities in the data to classify unseen FVs is a vital objective and intended to be realized by PBC. Yet, the creation of such patterns can be considered data-intensive such that both TDB and PDB are frequently penetrated by PBC. This situation is crucial and causes a lot of communication overhead when assuming that these three components are independent. From that perspective, it is favorable to bring the pattern building process close to the data. Hence, in-DB analytics are the logical choice but, unfortunately, existing solutions pointing in this direction are inappropriate and other methods worth to consider carry essential design flaws. For this reason, we devote this chapter to the development of a new algorithm capable to discover hidden patterns inside relational DB systems. This approach pursues an incremental learning strategy and is based on our in-DB VPRS model. Moreover, it is designed to run in nonstationary environments and attempts to address class imbalance implicitly, which are inherent problems of our target domain. In order to demonstrate its capabilities, we compare our algorithm with related work under different drift scenarios in terms of predictive performances, discovery-oriented aspects and time demands. Results of this general assessment show that our pattern builder is superior compared to state-of-the-art algorithms in various aspects.

This chapter is based on:

F. Beer, U. Bühler: "Learning adaptive decision rules inside relational database systems". In: *Proceedings of the 2nd International Symposium of Fuzzy and Rough Sets (ISFUROS 2017)*, pp. 41:1–41:12, 2017.

F. Beer, U. Bühler: "In-database rule learning under uncertainty: a variable precision rough set approach". In: R. Bello, R. Falcon, J. L. Verdegay (eds.) *Uncertainty Management with Fuzzy and Rough Sets - Recent Advances and Applications*, Studies in Fuzziness and Soft Computing, vol. 377, pp. 257–287, Springer, 2019.

9.1 Introduction

In order to materialize several desired features of the proposed HFIDS architecture, quantities of processed flows have to be retained. The reasons for this demand are twofold. On the one hand, historic data and relevant processing statistics are meaningful sources to build reports for AM to keep track of the overall system behavior. On the other hand, recently captured and analyzed FVs serve ADC to retrain its underlying model periodically. Consequentially, a requirement to serve these aspects is the integration of a suitable DB system maintaining relevant information, i.e. TDB. Closely connected to this data store is another integral part of the overall system architecture handling the automated generation of patterns in the form of decision rules. This subject is conducted by PBC and conceptually elaborated in this chapter. Primarily, it is intended to build, update or even delete patterns on the basis of FVs arriving successively to the system that either passed classification at PMC or ADC while persistently managed inside PDB. This way, PBC highly contributes to an increased transparency with respect to an operational standpoint. Yet, the extraction of patterns and their maintenance is data-intensive and, quite naturally, constitutes an frequent penetration of PDB and TDB due to heavy read and write operations, which evokes communication overhead in addition. Thus, it is a favorable mission to reduce these huge data movements among the three components. This can be achieved by exploiting the in-DB paradigm which we substantiated earlier. In this context, a further design consideration for PBC is concerned with the timely deployment of patterns. The argument behind this undertaking is that a rapid transformation of incoming FVs into decisive patterns certainly can improve hit rates at PMC such that classification of unseen data benefits enormously relaxing black box predictions at ADC. However, tackling this challenge is beyond the scope of conventional rule learners (e.g. AQ [56], PRISM [61], RIPPER [316]) as their basic learning assumption relies on complete training data availability right before the model building can commence. Even refining such a batch learning setup similar to the ideas pursued for ADC is conceptually discouraging because the inherent time gap between FV arrival and pattern deployment cannot be reduced properly. Moreover, new unintended complexity would be introduced if we would consider ensemble methods at this point requiring either a sophisticated fusion strategy for concurrent models or a general conflict management for matching but divergent patterns. Obviously, such considerations violate the prompt pattern building demand suggesting to learn in a rather incremental fashion. This is enabled by "incremental learning" referring to a type of ML algorithms following a continuous learning strategy by stepwise updating an already existing model (e.g. [408, 409, 410]). Beyond discussing the timely deployment of patterns, the appropriate handling of drifting concepts is perhaps more crucial. But even in that discipline, traditional rule inducers are not beneficial either as they presume data to be drawn from the same distribution. Thus, more promising rule-based approaches need to be considered known from recent stream mining literature (e.g. [98, 411, 412]). For the most part, they deal with incremental learning in nonstationary environments but carry other essential drawbacks in terms of scalability as a result of their stiff underlying architecture. Another preoccupation for PBC is that class labels of incoming FVs may not correspond to the true class necessarily as a result of a hidden noise process modifying the true label right before it is presented to the

learner (see [413]). This phenomenon is of high importance from the design perspective of PBC and is referred to as "imperfectly supervised learning" or "label noise" (e.g. [414, 415, 416]) because its input partially relies on predictions made by ADC. Thus, the arrival of potential wrong class labels can badly influence the generation of patterns due to FP and FN produced by the ensemble, which needs sound handling.

Based on the described benefits of in-DB processing, the exigency of incremental learning under nonstationary environments and the shortcomings of related alternatives, we picture a new incremental rule-based algorithm in this chapter, which sets the ground for PBC. It is termed "incremental in-DB rule inducer" (InDBR) and leverages our derived in-DB VPRS implementation (see Chapter 4). The motivation behind this choice is two-fold. On the one hand, the in-DB design not only reduces communication overhead, but yields a technical platform to overcome the deficits of existing approaches permitting to establish a scalable pattern building process. On the other hand, VPRSs are employed in particular because they can manage label noise as part of their intrinsic uncertainty framework. Additionally, variable precision supports smoothing minor data irregularities such that deviations between the actual model and observed data can be tolerated to some extent. This is realized by rendering certain and uncertain decision rules using a new "bottom-up" (or specific-to-general) rule induction approach, which successively generalizes specific rules fostering a prompt pattern building at the same time. Literature frequently endorses bottom-up learning as well. For instance, [100] see bottom-up learning better suited for incremental settings as opposed to "top-down" (or general-to-specific) approaches even though top-down rule inducers tend to produce more general patterns and less complex models. Another upside is attested by [417, 418] as a tighter connection to the data is established and so concrete explanations are given by patterns. Beside these benefits, the handling of data imbalance problems and concept drifts is a compulsory requirement. Therefore, an distinctive data buffer is devised, which is supported by a rigorous strategy to prune obsolete patterns respectively. Given these features as baseline, InDBR fulfills the required characteristics of PBC empowering a continuous pattern building in the context of flow-based intrusion detection that are meaningful and interpretable by operational staff. Especially because of these capabilities, we underline that InDBR is not limited to this target domain only. Beyond that, it can be applied to other disciplines serving general-purpose characteristics where in-DB pattern extraction is a desirable goal. This is verified empirically by assessing InDBR on various data sets from a quantitative and qualitative angle showcasing its advantages over related algorithms in several dimensions.

In the remainder of this chapter, we first contextualize closely related approaches (Section 9.2) followed by the introduction of InDBR as new in-DB pattern builder. In this light, patterns are examined from a formal perspective including elementary properties. Furthermore, a basic data structure is sketched to cache current examples, which is leveraged during the patterns building process. This conceptual perspective of InDBR is further concretized by illuminating important aspects of its implementation (Section 9.3). Based on this picture, InDBR is practically assessed in relation to state-of-the-art rule learning algorithms. Therefore, we frame an environment to conduct experiments, which is used to study predictive performances, discovery-oriented capabilities and time demands (Section 9.4). Lastly, we recap and discuss essential findings (Section 9.5).

9.2 Related Work

While we reviewed classic ASG solutions in Section 5.4 figuring out that these methods are rather inappropriate for our purpose, the focus of this section is on decision rule learning. In particular, we concentrate on closely related methods that are capable to learn incrementally and that are designed for nonstationary environments. One of the first approaches is a family of algorithms called FLORA [419] consisting of FLORA2, FLORA3 and FLORA4. The main idea behind FLORA2 is the utilization of an sliding window to frame most recent data for the rule induction process. This window is able to expand or contract dependent to the data context and, thus, enables the rule engine to adapt to drifts. FLORA3 is an extension of FLORA2 with a focus on reoccurring concepts. This is done by reconsidering useful rules after each learning cycle. To distinguish between concept drift and noise in the data, FLORA4 especially tracks the quality of rules through confidence intervals. AQ11-PM+WAH [420] is another method that is derived from the classic sequential covering algorithm AQ. It incorporates the adaptive window heuristic of [419]. Hence, it is comparable to FLORA2 from a predictive perspective. Yet, it maintains fewer examples over the course of learning. In this context, it is worth noting that neither AQ11-PM+WAH nor the FLORA algorithms are designed to process massive data streams that are potentially infinite. The FACIL algorithm [421] can be considered one of the first stream mining approaches. It pursues a bottom-up rule induction strategy and is explicitly built to mine numeric data. Its underlying model manages rules with a collection of associated examples permitting to store positive and negative examples, which are very close with one another, i.e. taking care of border examples. Hence, FACIL is able to store inconsistent rules, which is the core conception of this algorithm. Based on that, rules violating a user-specific minimum purity are replaced by new rules generated from corresponding examples. A different view is taken by VFDR [98]. It follows a top-down approach stepwise specializing existing rules and implicitly conforms to concept drifts. Inspired by the tree-based algorithm VFDT [422], its rule induction is guided by the "Hoeffding bound" [423]. It is utilized to determine when to specialize existing rules or when to induce new ones. An extension to VFDR is provided in [424] for multiclass problems by decomposing the task into two-class problems. To further improve its performance under changing conditions, VFDR is equipped with an explicit drift detector in [418] yielding an enhanced version of the algorithm called AVFDR. In the context of stream-based learning, a frequent demand is the ability to classify arriving examples at any-time, i.e. the "anytime property" (e.g. [425, 426]). To address this requirement, VFDR and its extensions incorporate NB functionality to rate uncovered examples. An approach explicitly relaxing this property is eRules [411], which exploits the well-known rule learning algorithm PRISM. To enable PRISM with stream mining capabilities, eRules simply buffers unclassifiable examples that arrive over time and triggers PRISM once the cache exceeds. As a result, classification is abstained when no appropriate rule exists or the algorithm is uncertain. Its successor is called G-eRules [412] and is introduced due to the poor performance of eRules when confronted with continuous data. The latest evolution of eRules is termed Hoeffding Rules [93]. It is quite similar to G-eRules but incorporates the Hoeffding bound as a statistical measure to determine the number of examples required to stimulate the production of new decision rules. The most recent bottom-up

rule inducer for nonstationary environments is RILL [427]. It is an any-time algorithm based on distance measures to find nearest rules in order to classify arriving examples that are not covered. Additionally, it employs intensive pruning operations to keep most essential information only. This is an important step to reduce costs of performed distance measurements. Note, further details on a subset of the mentioned incremental rule learners are surveyed in [428].

In contrast to the presented approaches, our proposed incremental rule-based algorithm leverages VPRSs and was originally introduced in [429, 430]. It is designed for in-DB applications and, thus, is supported by very efficient query plans running in parallel. Additionally, it adopts the idea of [93, 411, 412] and other authors to relax the any-time property. This is an essential factor for real-world scenarios requiring reliable predictions that are interpretable by decision-makers. Contemplating existing full-featured in-DB RST implementations as given by [173] or [176] is not meaningful at this stage because they can neither cope with nonstationary environments nor handle uncertainty. In particular, the latter point strongly conflicts with the intrinsic virtue of RST and VPRSs respectively and refuses their usage when data sources are imperfect which we already discussed in Section 4.5.1 in the context of FS.

9.3 Incremental In-Database Rule Inducing

In this section, we introduce InDBR as new in-DB rule learner by formally defining patterns with associated properties including a method to induce them using our VPRS model (Section 9.3.1). Furthermore, a mechanism is sketched constituting InDBR's perception on its outer environment (Section 9.3.2). Based on that, we outline operational principles of it from a conceptual standpoint (Section 9.3.3), which is leveraged next to highlight important aspects of its SQL implementation (Section 9.3.4).

9.3.1 Knowledge Representation

During the course of this work, we came across the term "pattern" several times and provided rather informal descriptions. In this section, patterns are illuminated from a theoretical perspective outlining inherent properties and a way to induce them using our in-DB VPRS model. In this respect, we restrict ourselves to learn patterns in propositional form as it offers a simple and lightweight representation of knowledge to solve classification tasks. This design choice entails that no prior knowledge is assumed and that no relations among attributes can be expressed as opposed to typical relational learning systems such as given in [431, 432]. Instead, pattern induction exclusively relies on training examples with defined attribute schema and decision class that become available over time constituting an incremental supervised learning setup. Based on these circumstances, the pattern description language can be formally framed by Definition 9.1.

Definition 9.1: *Given a domain of interest described by examples with condition attributes* $A = \{a_1, ..., a_m\}, m \in \mathbb{N}$ *and the decision or class attribute* $D = \{d\}$*, a pattern*

is a propositional formula

$$P \Rightarrow Q \tag{9.1}$$

with premise P and conclusion Q. P constitutes a conjunction of logical tests represented by literals of the form $(b = v)$ implying Q given by $(d = c)$, which, combined, leads to

$$r_{B,D} := \Big(\bigwedge_{b \in B} (b = v) \Big) \Rightarrow (d = c) \tag{9.2}$$

where $B \subseteq A, v \in V_b, c \in V_d$ with V_b and V_d are corresponding value ranges of attributes b and d. Thus, a pattern $r_{B,D}$ is a logical expression describing identified regularities in the data entailing a classification, i.e. a decision rule.

Due to the rich history of decision rule learning, premise P and conclusion Q are often described by other synonyms. While P is also referred to the terms "descriptor" or "condition", Q frequently is designated as "consequent". As such, a rule or pattern can intuitively be understood by the assertion: "if condition then consequent" (see Section 2.2.2). This way, a rule is applicable to an example if its condition part holds true, which suggests the corresponding consequent. In this regard, a rule is also said to "cover", "match" or "hit" a particular example. Reviewing Definition 9.1 further discloses two constraints, i.e. literals support the equality operator and a single decision attribute only. The former can be explained based on the fact that the current version of our incremental rule inducer solely operates on discrete data whereas the latter is the result of a simplification satisfying our domain of interest and many other classification tasks. For this reason, we restrain further formalisms consequently to one decision attribute w.l.o.g. and take advantage of this design choice streamlining the notation of a rule. We use r_B as shorthand for a concrete pattern $r_{B,D}$ or simply write r for the sake of convenience.

Having established the definition of patterns paves the way to introduce further characteristics, which are important to comprehend an intuition of InDBR's processing. The first property is concerned with the "complexity" or "length" of patterns provided by Definition 9.2.

Definition 9.2: *Given a decision rule r_B, its length can be described by the number of literals or by the cardinality of the attributes equivalently enclosed by its descriptor. Formally, this measure can be expressed by*

$$len(r_B) := |B| \; . \tag{9.3}$$

Earlier, we mentioned the applicability of a rule to an example. This terminology is somewhat misleading because a rule can match multiple examples at the same time. This directs us to the "coverage" of a rule as a composition of two disjoint sets defined by the following statements.

Definition 9.3: *The positive coverage of a decision rule r is denoted by $cov_p(r)$ and encompasses examples satisfying r's condition part and its consequent. Thus, $cov_p(r)$ is the set of examples that are correctly predicted by r. Conversely, the negative coverage $cov_n(r)$ is the set of examples that are covered by r but its conclusion fails constituting wrong predictions. As a result, we obtain the complete coverage of r by the union of both*

posed sets, i.e.

$$cov(r) := cov_p(r) \cup cov_n(r) \ . \tag{9.4}$$

The coverage measures as given by Definition 9.3 are typically specified based on all available training examples. As opposed to a classic ML setup where this data population is known by definition, its establishment in an incremental setting requires thorough deliberation. Therefore, we defer further discussions on this subject to Section 9.3.2. Instead, we simply keep in mind that the introduced coverage measures are implicitly related to a defined collection of examples for the time being. The next important characteristic that builds on top of the coverage is concerned with the quantification of a pattern's quality using Definition 9.4.

Definition 9.4: *The error produced by a decision rule r is defined by*

$$\delta(r) := \begin{cases} 1 - \frac{|cov_p(r)|}{|cov(r)|} & , \textit{ if } cov(r) \neq \emptyset \\ 0 & , \textit{otherwise} \end{cases} \ , \tag{9.5}$$

while the purity or the precision of r essentially expresses the opposite of the error. Therefore, it can be computed by $1 - \delta(r)$.

Note, δ is closely connected to the misclassification rate of Definition 3.11 and together with the purity a quite intuitive measure. Other quality metrics are established in rule learning literature as well. Some non-exhaustive lists can be found for example in [100, 433, 434]. Given these definitions, we are able to compare patterns along the properties: purity, coverage and length. Particularly for the latter two cases, a strong relation can be inferred. On the one hand, complex patterns are close to describe an example and therefore have a tendency to produce a small coverage. On the other hand, patterns with a short length mainly provide a higher coverage. This supports the introduction of the term "rule generality" as essential concept to predict previously unseen examples given through Definition 9.5.

Definition 9.5: *A rule $\hat{r}$ is more general or simpler than a rule r in the proper sense if and only if (i) both rules carry the same consequent and (ii) $cov(\hat{r}) \supset cov(r)$. In this context, r is also said to be more specific than $\hat{r}$ or r refines $\hat{r}$.*

Thus, the idealized view of rule learning is to build a hypothesis consisting of a set of rules (see Section 2.2.2) with a high overall coverage and low error. This in turn infers that a rule set should be "complete" and "consistent" with available training data and yet capable to correctly classify unseen examples. Qualitatively, this idea is presented in Figure 9.1 with four different hypotheses describing the positive class of a binary classification problem. In Section 9.3.3, we revisit this subject and see that in real-world applications it is rather undesired to completely pursue the learning of a perfect rule-based model.

While the entire induction process is illuminated later in this chapter, some preliminary work is still required upfront with respect to the generation of patterns based on examples which we anticipate in the remainder of this section. In particular, this task is guided by key concepts of VPRSs, i.e. the β-region $\mathcal{L}$ and $\mathcal{B}$ given through Theorem 4.2. Both relational expressions, however, suppress the decision attribute, which can be solved by

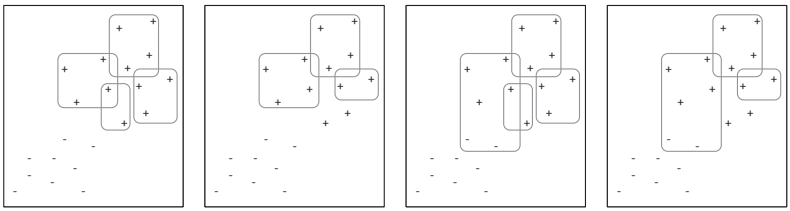

FIGURE 9.1: Different situations of a rule-based model explaining the positive class which is (a) consistent and complete, (b) consistent and incomplete, (c) inconsistent and complete and (d) inconsistent and incomplete (adapted from [100])

either (i) modifying the queries to output this feature in addition without great impact or by (ii) introducing supplementary steps to finally synthesize patterns. Certainly, (i) is more appealing performance-wise but out of illustration purpose we apply (ii) and keep the original queries untouched. Given an input table $T_{\langle A,\{d\}\rangle}$ containing examples, this option can be computed using the following prototypical relational expression

$$\omega(\mathcal{Q}^{\beta}_{B,\{d\}}(T)) := \tilde{\mathcal{G}}^{\{c_t\}\cup B\cup\{d\}}_{c_p,\{c_t\}\cup B\cup\{d\}}(\mathcal{Q}^{\beta}_{B,\{d\}}(T) \bowtie T) \tag{9.6}$$

where $\mathcal{Q}$ either stands for $\mathcal{L}$ or $\mathcal{B}$ with the condition features $B \subseteq A$ and the tolerated error $\beta \in [0, 0.5)$. Hence, ω extracts decision rules r controlled by B and β. In addition to descriptor and conclusion, the final schema of ω also contains statistics of each particular r through the attributes c_p and c_t, which correspond to $cov_p(r)$ and $cov(r)$ respectively. Considering the extraction of certain rules via $\omega(\mathcal{L})$, several rules may exist with different decisions but same descriptor as a result of the admissible error β. This can cause inconsistencies even though one of the conflicting rules positively covers the majority of examples. To counteract such situations, we are only interested in those rules r exposed by $\omega(\mathcal{L})$ maximizing $cov_p(r)$ per conflict and in those not disputing. Conversely, rules produced by $\omega(\mathcal{B})$ are quite uncertain by definition but may be of interest for future use. Thus, InDBR is able to maintain both types of rules, which is further detailed in Section 9.3.3.

9.3.2 Partial Memory

In nonstationary environments, sliding windows based on first-in/first-out (FIFO) principles are common practice to buffer incoming examples, which in turn are exploited to rebuild or update an existing predictive model. One of the first approaches using this kind of data management is FLORA2 and since then several other representative window-based techniques emerged (e.g. [435, 436, 437]). By design, this type of data structure strongly biases towards data recency. This is desirable when working with changing conditions in the data but despite their simplicity and popularity they tend to forget rare occurrences[1] rather quick [438]. This is a general problem observable in concept drift and stream mining literature as most approaches neglect class imbalance

[1] We are not referring to outliers here but to rarity in the sense of class imbalance (see Section 8.1).

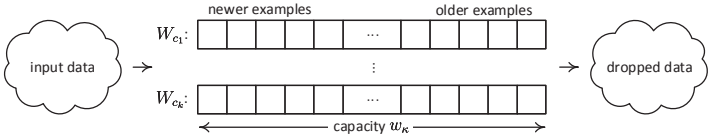

FIGURE 9.2: Proposed partial memory consisting of k sliding windows each with a capacity of w_κ

(see [439, 440]), which affects our domain of interest as well as several other applications. To counter this finding, we propose a modification of conventional window-based methods by utilizing multiple sliding windows instead. In this respect, we are aware of few other significant works that rely on more than one window (e.g. [121, 441, 442]). [121] employs three competing windows with varying size, which provide different resolution levels to interpret drift scenarios. [441] incorporates ensemble learning with a double window mechanism and demonstrates its superior over a single sliding window. In the context of "multi-label classification", [442] proposes a method that maintains two windows for each label in text mining applications. In contrast to these techniques, our sliding window approach is explicitly designed for multiclass classification and attempts to counter class imbalance problems, which is briefly outlined in the remainder of this section.

We already reported about oversampling and undersampling as two state-of-the-art methods to equalize skewed class distributions (see Section 8.3.1). This simple idea, often utilized for conventional learners that bias towards the majority classes, can even be ported to the context of window-based approaches under the assumption that examples of minority classes are kept longer in the window compared to those of the majority classes. A similar effect can be reached quite naturally when considering one window per class, which leads to the basic idea behind our approach. Instead of maintaining a single window, our method employs k independent sliding windows $W = \{W_{c_1}, ..., W_{c_k}\}$ where $k \geq 2$ is the number of concepts we pursue to learn modeled by the class attribute d with $V_d = \{c_1, ..., c_k\}$. In this regard, each window is equipped with a predefined capacity $w_\kappa \in \mathbb{N}$ constraining the number of permitted examples[2]. To understand its principles, let us assume an empty collection of windows with $W_{c_i} = \emptyset, 1 \leq i \leq k$ as starting point. Thus, arriving data can be added to W such that examples comprising class c_i are associated with window W_{c_i} as long as $|W_{c_i}| \leq w_\kappa$ is fulfilled. Consequently, this basic policy infers that no further examples can be added to W_{c_i} after the maximum capacity is reached. Hence, a "rebalancing operation" is introduced permitting a continuous acceptance of arriving examples. This can be achieved by simply removing the oldest example in W_{c_i} in order to create sufficient space for a new example in situations where $|W_{c_i}| = w_\kappa$. An illustration of this approach is sketched in Figure 9.2 revealing a data structure maintaining $k \cdot w_\kappa$ examples in worst-case, which corresponds to the functioning of k independent queues intuitively. In this way, the union of all W_{c_i} can be considered as "partial memory" of InDBR comprising only a subset of all training data, which states the baseline to compute coverages and statistics for a proper pattern induction process under drift in multiclass and class imbalance settings.

[2] This limitation is required because we cannot keep all of the examples in a setup expected to produce data continuously at a potentially infinite scale.

9.3.3 Induction Process

Taking the previous considerations into account, the incremental in-DB rule learning approach is outlined in this section. First of all, we briefly summarize its main idea consisting of four notable building blocks. Secondly, these blocks are described in greater detail in order to get a complete picture of its main operating principles with a focus on training and generalization procedures.

As incremental algorithm, InDBR learns and adapts continuously based on labeled training examples that arrive over time. These are retained in a bounded buffer serving as a data source to update its underlying model or to create new patterns in case parts of the examples are still unexplainable by the current hypothesis. To attain these objectives, its existing rule set is generalized in the first place and remaining examples still not covered after this action are transformed into most specific rules. For that purpose, several key aspects of VPRSs are employed rendering a novel bottom-up induction approach that ensures a high coverage from the current processing viewpoint. Ultimately, the predictive model quality is maintained by pruning obsolete patterns. Under these directives, the training cycle of InDBR not only permits the rule set to evolve incrementally as new data arrive but it also keeps complexity low by consistently purging both training examples and decision rules concentrating on the most recent input. This constitutes an important characteristic to promptly response to changing conditions that are expected to occur due to the nonstationary nature of the data. The internals of this approach are highlighted in Algorithm 9.1, which illustrates the essential training steps given in the form of pseudocode[3]. To further detail the functionality, we categorize this training procedure into four main building blocks that are outlined as follows:

(i) Consolidate incoming data and statistics (line 2 to 3)

(ii) Generalize existing rule-based model (line 4)

(iii) Extract new most specific rules (line 5 to 8)

(iv) Manage rule aging and retirement (line 9 to 11)

The initial step (i) of InDBR's processing refers to the appropriate integration of incoming examples and to update affected pattern statistics respectively. In this way, a generic solution is provided in terms of data reception supporting two different modes, i.e. example-by-example or batch-by-batch processing, both symbolized by input T. The first mode enables comparability with related work from stream mining literature treating arriving examples in sequential order for the most part. However, its downside is a rather poor processing when applied to a DB system. As underlying platform of InDBR, these systems are much more efficient when confronted with batches of data permitting the utilization of parallel query plans, which justifies the second mode. In this regard, InDBR's input can be adapted to different scenarios accordingly using parameter u to limit the size of T. The management of incoming training examples provided by T is undertaken by a collection of sliding windows $W = \{W_{c_1}, ..., W_{c_k}\}$. As outlined

[3] In this context, pseudocode preserves readability as opposed to cumbersome and highly technical SQL statements. However, one can verify its translation to the domain of DB system can be carried out straightforwardly as discussed in Section 9.3.4.

Predefined settings: $A = \{a_1, ..., a_m\}, m \in \mathbb{N}$ (condition attributes), $D = \{d\}$ (decision attribute), $\alpha_c \in \mathbb{N}$ (maximum rule age $\forall c \in V_d$), $\beta \in [0, 0.5)$ (tolerated error), $u \in \mathbb{N}$ (maximum size of input batch), $t \in \mathbb{N}_0$ (leap constant), $g : \mathbb{N} \to [0, 1]$ (mapping to determine the percentage of rules to generalize), R (rule set), W (partial memory)
Input: T (batch of examples with conditions A, decision D of size u)
Output: R

BEGIN
  $W \leftarrow W \cup T$ /*append T and rebalance W in case its buffer exceeds*/
  update statistics of R with respect to W
  $R \leftarrow \textsc{GenRuleSet}(\beta, t, g, R, W)$ /*generalize rule set */
  $O \leftarrow T \setminus \bigcup_{r \in R} cov(r)$ /*determine uncovered examples*/
  $L \leftarrow \omega(\mathcal{L}^{\beta}_{A,\{d\}}(O))$ /*all new active rules*/
  $B \leftarrow \omega(\mathcal{B}^{\beta}_{A,\{d\}}(O))$ /*all new inactive rules*/
  $R \leftarrow R \cup L \cup B$ /*append new rules to R*/
  $P \leftarrow \{r \in R \mid cov_p(r) \cap T \neq \emptyset \wedge \delta(r) \leq \beta\}$ /*positive coverage of batch*/
  $\alpha_r \leftarrow \begin{cases} 0 & \text{, if } r \in P \cup L \cup B \\ \alpha_r + 1 & \text{, otherwise} \end{cases}, \forall r \in R$ /*maintaining rule aging*/
  $R \leftarrow R \setminus \{r_{B,\{d\}} \in R \mid \alpha_r \geq \alpha_c \text{ with } (d = c)\}$ /*retire antiquated rules*/
END

Algorithm 9.1: Training procedure of InDBR to induce patterns incrementally

in the previous Section 9.3.2, this data structure is of fundamental importance because it acts as partial memory and reflects InDBR's perception on the real world in order to infer new patterns or to generalize the existing rule set R consequentially. In particular, W's perimeter to a total capacity of $k \cdot w_\kappa$ addresses three notable points, i.e. enabling a constant management of input data and focusing on most recent examples grouped by class, which in turn supplies active support for the handling of data imbalance and concept drift. Yet, one has to be cautious due to this restriction because there is a direct relation between the sizing of T and W relevant for the mentioned batch mode to evade potential data overwriting. To contextualize and configure T and W adequately, we can estimate the proportion of exchanged examples for each single class window $W_{c_i}, 1 \leq i \leq k$ during training, which roughly corresponds to u/k looking at a perfectly balanced classification problem. In cases where a highly imbalanced class distribution is present, this proportion increases to u only pertaining a single class window. Putting these observations into perspective yields upper limits for u preventing to completely overwhelm W with data from T, i.e. $u \leq k \cdot w_\kappa$ and $u \leq w_\kappa$ for both balanced and imbalanced settings. Based on those two extremes, a convenient setup for u and w_κ is given by $u \ll w_\kappa$ for the purpose of reaching an acceptable percentage of examples still remaining in W for upcoming training rounds. Beside these data management considerations, changes to W's content become prevalent over time as a matter of fact affecting R to age inevitably. Hence, underlying statistics of R need adjustment eventually, which is revised in the initial step (i) as well.

The next building block (ii) is concerned with the generalization of existing patterns in order to extend earlier gathered knowledge based on new training examples collected in step (i) and to increase its predictive capabilities ultimately. While these objectives seem to be obvious and are partially shared among other incremental rule learning methods, notable differences can be highlighted to pursue this target. As opposed to other works, the heart of our generalization approach not only relies on the key

conception to seek for new appropriate decision rules in the search space that simplify an already existing pattern. Moreover, the rule engine takes advantage of this gained knowledge, i.e. the generalized attribute sets. Leveraging this knowledge supports us to infer other previously unknown and significant patterns relying on the same premise, which increase the chance to constitute a higher inductive leap. This is accomplished using the β-positive region with minor computational overhead due to our in-DB VPRS model. Distinguished by these characteristics, the rule generalization is depicted in Algorithm 9.2 and further detailed in the remainder of this paragraph. It starts off with the partitioning of R according to rule length j inducing disjoint subsets $R_j \subseteq R, 1 \leq j \leq m$. Based on this resulting structure, we are able to explore the rule space systematically by traversing the partition in ascending order to generalize proportions $G \subseteq R_j$. In this context, it should be denoted that simplifications on R_1 (i.e. the set of patterns containing only a single literal in the condition part) are neglected due to practical reasons[4]. Hence, the iterative processing starts at $j = 2$ and extracts new decision rules straightforwardly using "dropping conditions" [57]. In essence, this well-established induction technique infers new patterns by stepwise removing attributes from a reference parent rule until a predefined stopping criterion is met. In our case, only one simplification step is permitted for a rule $r_B \in G$ in a specific training round such that child rules $\hat{r}_{B\setminus\{b\}}$ can emerge with $b \in B, len(\hat{r}_{B\setminus\{b\}})+1 = len(r_B) = j$ and $cov(\hat{r}_{B\setminus\{b\}}) \supset cov(r_B)$. This designates a "minimal generalization operation" [100] because we obtain for all generalizations $\hat{r} : cov(\hat{r}) \supset cov(r) \wedge \nexists \bar{r} : cov(\hat{r}) \supset cov(\bar{r}) \supset cov(r)$. Further constraining the search, a heuristic is employed to select rules with minimal error $\delta \leq \beta$. By means of this methodology, most certain and meaningful patterns are unveiled among the generated candidates constituting a one-step bottom-up hill climbing approach in the proper sense, which is outlined in Algorithm 9.3. Recalling that this procedure is executed for every $r \in G$, the cardinality of G is crucial with respect to the trade-off between inductive leap and computational effort. Therefore, a mechanism is introduced that is capable to manage the generalization behavior. It can be adjusted application-specific and it is realized by a prototypical partial mapping g, which determines the percentage of rules to simplify per iteration j with $|G| = \lceil g(j) \cdot |R_j| \rceil$. While this approach produces truly more general rules, the obtained knowledge can be recycled to seek further generalizations for parts of the partition not considered yet. Specifically, we preserve the attribute sets $S \subseteq \mathcal{P}(A)$ from rules compiled previously with length $|B| = j - 1, \forall B \in S$. These are utilized as seeds to inquire simplifications for rules in $R_o, o > j$ by computing $\omega(\mathcal{L})$ disregarding examples in W already covered by former generalization steps. As a result, new patterns $\tilde{r} \in \tilde{R}$ are derived. Obviously, these could directly substitute their more specific predecessors but there is a certain tenancy for overgeneralization considering the potential high simplification gain acquired, i.e. $o \gg j$. Therefore, a leap constant t is proposed that permits such abrupt generalizations only if $\tilde{r}$ provides sufficient evidence with respect to W and its parent rule r such that $|cov(\tilde{r})| \geq |cov(r)| + t$. Note, further demands on $\delta(\tilde{r})$ can be omitted at this stage because the generation of $\tilde{r}$ is inherently constrained by the β-positive region. Having discussed the main stages of the generalization over iteration j, the processing continuous with $j + 1$ until all granularity levels of the rule space are examined. Subsequently, a final cleanup activity is performed dropping undesired simplifications. This is an

[4] In fact, such rules can also be dumbed down to a rule covering every example, i.e. the universal rule $r_\emptyset$. Nonetheless, this consideration is of high theoretic nature and therefore ignored subsequently.

Input: β, t, g, R, W
Output: R (generalized rule set)

BEGIN
  $PR \leftarrow \{R_j \subseteq R \mid j = 1, ..., m\}$ with $R_j = \{r \in R \mid len(r) = j\}$ /*partitioning rule set according to the rule length*/
  $\hat{C}_r \leftarrow \begin{cases} \{r\} & , \text{if } r \in R_1 \\ \emptyset & , \text{otherwise} \end{cases}, \forall r \in R$ /*initialize candidate sets of generalized rules*/
  $\check{C} \leftarrow \emptyset$ /*initialize set of best rule candidates*/
  FOR $j \leftarrow 2, ..., m$ **LOOP**
    $\hat{C}_r \leftarrow \textsc{GenRule}(r, \beta), \forall r \in G \subseteq R_j$ where the proportion of rules G is randomly selected using $g(j)$ for $\hat{C}_r = \emptyset$
    $S \leftarrow$ collection of attribute sets for those $\hat{C}_r$ compiled in the previous step
    $\tilde{R} \leftarrow \bigcup_{B \in S} \omega(\mathcal{L}^{\beta}_{B,\{d\}}(W \setminus \bigcup_{\hat{C}_r \neq \emptyset} \bigcap_{\hat{r} \in \hat{C}_r} cov(\hat{r})))$ /*induce new rules of length $j-1$ from W without examples certainly covered by all $\hat{C}_r \neq \emptyset$*/
    $\hat{C}_r \leftarrow \hat{C}_r \cup \{\tilde{r}\}, \forall \tilde{r} \in \tilde{R}, r \in R_o$ where $o > j$ and an increased coverage is obtained such that $cov(\tilde{r}) \supseteq cov(r)$ and $|cov(\tilde{r})| \geq |cov(r)| + t$
  END LOOP
  $\check{C} \leftarrow \check{C} \cup \{\check{r}\}$ where $\check{r}$ is the best rule in $\hat{C}_r$ maximizing $1 - \delta(\hat{r})$ and $cov(\hat{r})$ setting $\alpha_{\hat{r}} = \min(\alpha_{\hat{r}}), \forall \hat{r} \in \hat{C}_r, \forall \hat{C}_r$
  $R \leftarrow R \setminus \{r \in R \mid \exists \hat{C}_r \neq \emptyset\}$ /*remove all obsolete rules from the rule set*/
  $R \leftarrow R \cup \check{C}$ /*append new more general rules to rule set*/
END

Algorithm 9.2: Bottom-up rule set generalization procedure GenRuleSet

Input: r_B with $B \subseteq A, \beta$
Output: C (generalized rule candidates)

BEGIN
  $C \leftarrow \emptyset$
  FOR $b \in B$ **LOOP**
    $\hat{r} \leftarrow r_{B \setminus \{b\}}$
    IF $cov(\hat{r}) \supset cov(r) \wedge \delta(\hat{r}) \leq \beta$ **THEN**
      $C \leftarrow C \cup \{\hat{r}\}$
    END IF
  END LOOP
  $C \leftarrow \arg\min_{\hat{r} \in C} \delta(\hat{r})$
END

Algorithm 9.3: Procedure GenRule to generalize a pattern by dropping conditions

obligatory step because parent rules r can exhibit multiple child rules or share the same generalizations $\hat{r}$, i.e. siblings or duplicates. Thus, the pattern with maximum purity $1-\delta(\hat{r})$ and highest coverage $cov(\hat{r})$ is chosen among those siblings $\hat{r}$ that emerged from r indicated by $\check{r}$[5]. In this respect, those $\check{r}$ entail most promising child patterns, which are maintained in $\check{C}$, while taking care of duplicate elimination consequently. Ultimately, all predecessor rules in R are substituted by their obtained successor patterns in $\check{C}$ concluding the bottom-up generalization procedure.

Arguably, learning a consistent and complete hypothesis based on a set of rules is desirable reviewing Figure 9.1. This idealized standpoint, however, does not hold when the data generating process is imperfect and provided class labels are erroneous. In fact, rule learning in the presence of noise is a serious problem because it is prone to overfitting and tends to explain such random errors in the data rather than ignoring them (e.g.

[5] In case of a tie, we simply pick the first pattern among the siblings that comes along.

[100, 443]). For this purpose, β is employed to regulate δ, and we already perceived its usage on the generalization of existing rules outlined in the previous step (ii). These are by no means perfect but showcase confident patterns from a VPRS point of view. It can also be valuable to synthesize unreliable patterns with δ beyond β following the traditions of RST and extensions. Patterns of this kind contribute to the approximation of concepts by identifying regions in the data that are currently uncertain suggesting to abstain or to defer a decision until further evidence is present (see [444]). Additionally, these characteristics can support decision-makers to uncover and understand inherent quality issues or irregularities in data. Taking advantage of these insights, we equally make use of decision rules comprising an error in the range of $\beta < \delta < 1 - \beta$. Clearly, such rules cannot be used to predict incoming examples. Nevertheless, they might be valuable in future situations as data evolve. As a result, both types of rules are extracted in step (iii) based on those examples $O \subseteq T$ still not covered by the existing rule set. Therefore, $\omega(\mathcal{L})$ and $\omega(\mathcal{B})$ are applied on the entire feature set A to materialize most specific certain and uncertain decision rules, which are ultimately appended to R as part of InDBR's bottom-up rule induction approach.

The final step (iv) is concerned with some housekeeping on R by removing obsolete rules. This is carried out by the notion of "rule aging", which follows a straightforward approach. To exemplify its principles, we have to be aware that each pattern $r \in R$ has an age $\alpha_r \in \mathbb{N}_0$ initialized with $\alpha_r = 0$. Each time Algorithm 9.1 is called to train based on new incoming examples, existing rules in R undergo a process of stringent linear decay. This involves all uncertain rules by purpose. Certain rules are impacted as well in case they are unable to correctly classify at least a single example in T. The rule age of all other decision rules (i.e. most certain patterns positively matching input T) is reset consequently to their initial age. This way, a rule becomes obsolete and is removed from R if its maximum life span exceeds. This limit is defined by α_c and controlled per class c. Thus, the aging is adjustable depending on the current application and classification problems. While balanced tasks typically embody a constant maximum age along the class distribution, largely imbalanced settings often prefer a higher maximum age for rules supporting the minority classes and a proportionally shorter life span for rules of the majority classes. Reviewing this aging model from another angle, it can be understood as a plain competition implementing a kind of gradual forgetting. Initially, each created rule is equally treated per class irrespective of its current purity. Over the course of the training, their chance of being hit by arriving examples grows as a result of the bottom-up induction. InDBR only supports this evolvement within the bounds of α_c, i.e. patterns must enhance their reliability and reflect current structures in the data to evade the retirement process. As to other learning approaches, this kind of rigorous pruning is an important task to keep the model complexity low, which infers both a relaxation of computational demands and a straightening of the rule set at the same time.

Beside the pattern induction, the second integral part of InDBR is the classification of previously unseen examples with meaningful patterns. When it comes to this particular task, rule learning oftentimes sustain the inability to cover the entire data space, i.e. they have a tendency to produce incomplete hypotheses, as opposed to other ML algorithms such as most DTs and NB models. This characteristic can be crucial and contradicts

with strict demands, where the learner in charge should be able to predict at any time (see Section 9.2). To compensate this intrinsic problem, rule learners frequently utilize specific strategies such as the introduction of default rules or the orchestration of an additional predictor with any-time properties, which is trained in parallel. Despite these counteractions, they are still unfavorable for certain real-world problems. Suppose we are interested in quality predictions and these should be reproducible and explainable for decision-makers, neither of these points provide a satisfying solution. While the incorporation of default rules typically represent nothing more but the class distribution with very few explanatory power (e.g. predicting the label of the majority class), the embedding of an additional any-time ML model, again, turns the transparent rule-based approach into a black box predictor. This is particularly true in our domain of interest when it comes to alarming and no adequate pattern is in place guiding necessary follow-up activities for operational staff to reestablish the integrity of the network infrastructure under inspection. Thus, we conclude for such critical applications that the rule engine should fire only when it is most certain, i.e. quality and unambiguous rules exists. InDBR addresses these concerns by abstaining classification explicitly in cases where an adequate rule is absent or it is uncertain about a decision. This idea of skipping a doubtful classification rather than risking a potential wrong prediction is in line with the opinion of other authors (e.g. [93, 411, 412]). For that reason, InDBR's classification is only based on most certain rules by means of VPRSs. We refer to these rules as "active rule set", which can be formally described by

$$R_+ = \{r \in R \mid \delta(r) \leq \beta\} \tag{9.7}$$

showcasing the utilization of those rules in the rule set that are equal or below the tolerated error β. All other rules are denoted as "inactive rules" and composed by the set

$$R_- = R \setminus R_+ \tag{9.8}$$

consequently. Obviously, these are irrelevant for current classifications because of their high error but might be an interesting prospect. Particularly, this is the case for an eligible generalization in upcoming trainings or due to reappearing structures anticipated properly.

9.3.4 Implementation Aspects

The previous sections disclosed the principles of InDBR from a rather conceptual standpoint outlining main ideas and the pattern building process respectively. In this section, we devote our focus to its general-purpose implementation using SQL as primary language. However, a complete depiction is very technical and rather extensive and so we limit our attention to four key aspects providing an intuition of the entire implementation listed as follows:

- Main table schemas and representation of decision rules
- Population and rebalancing of the partial memory
- Maintaining rule coverage and updating statistics
- Implementation of the dropping condition heuristic

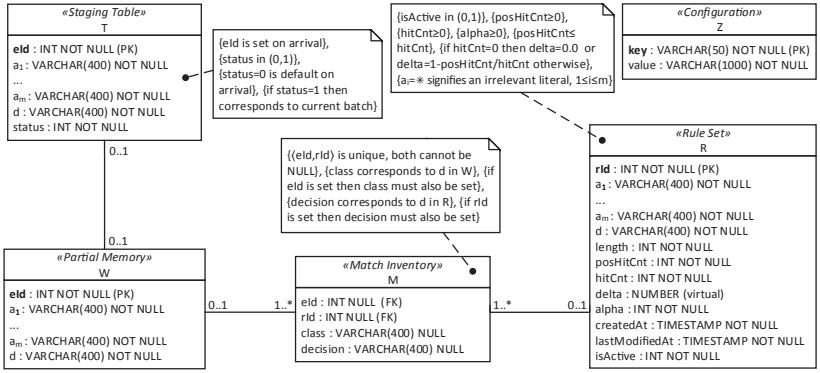

FIGURE 9.3: Entity-relationship model of InDBR representing input data, patterns and their relations along with constraints and configurations

Implementing incremental learning tasks inside a DB system requires appropriate data management given by a collection of concrete and exclusive tables in order to control input data, model, states and configuration settings consequently. In the context of InDBR, these were partially introduced in the previous section but are further enhanced herein for practical reasons. Summarized, five resident tables can be identified, i.e. the rule set R, the partial memory W, the staging table T, the rule matching inventory M and the configuration table Z. While T simply functions as data container for external data sources in this general-purpose implementation filling W with batches during training, M maintains the mapping between examples in W and patterns in R. Finally, Z is introduced to persist important configuration settings (e.g. A, D, β, u and t) in a key-value format. All of those briefly highlighted tables pose the key data structure of InDBR and are outlined as "entity-relationship model" in Figure 9.3. From a structural perspective, T, W and R follow a flexible condition schema with respect to $A = \{a_1, ..., a_m\}$ and $D = \{d\}$, which is dependently set and fixed according to the underlying structure of the data. At first glance, this seems to be a straightforward method modeling incoming examples because they completely fit the defined schema. Yet, proceeding this way lacks a suitable representation for patterns, whose length can vary considerably. In particular, this poses an issue when it comes to the efficient implementation of comparison operations using SQL as given through Definition 9.5. In this case, we have to compare R and W based on a predicate that is valid for both record types in order to seek more general patterns but a rule descriptor and an example's condition features might not correspond necessarily. Hence, modeling these dynamics paired with relational processing aspects require a unified comparison approach. This is achieved by introducing the "$*$-symbol" for rules with a length smaller than the number of all available condition features. It typifies the absence of literals in the rule descriptor, while maintaining the entire set of condition attributes for a pattern. In this way, we are able to treat both patterns and examples uniformly in a fixed table structure, which is exploited extensively in the rest of this section.

The staging table T essentially acts as interface for external sources permitting to place new training data in a concurrent fashion. During the continuous learning process, this

intermediate buffer is utilized to populate proportions of it to W with batches of size u or $|T|$, whichever is smaller. In this context, the population is based on a FIFO principle controlled by the attribute *eId*. It represents a sequential number automatically assigned to each example of T according on their arrival. Additionally, attribute *status* is used to mark those examples in T currently populated to W. Given this mechanism, the data populating process can be described in four steps: (i) we flag the oldest $\min(u, |T|)$ examples in T and (ii) copy them over to W. However, loading data into W might violate the capacity w_κ of certain class windows W_{c_i} requiring appropriate alignment. (iii) This is handled by the mentioned rebalancing operation of W. In essence, it observes each W_{c_i} and literally computes the amount of overhanging examples with respect to w_κ using $\lambda(c_i) = \max(|W_{c_i}| - w_\kappa, 0)$. Based on these obtained proportions, the oldest examples are removed from each W_{c_i} according to attribute *eId*, which reestablishes the expected bounds and behavior of W. The SQL implementation of this rebalacing operation is outlined in Listing 9.1. (iv) To complete the population activities, we can simply delete the batch of examples in the staging table that were marked earlier. Nonetheless, this action is deferred to the end of each training phase because this subset of records is reused subsequently (see line 5 and 9 in Algorithm 9.1). At this stage, it should be stated that the data population as well as the rebalancing involve sorting operations, which are typically undesired out of performance reasons. Concerns in this direction can be disregarded, though, when utilizing appropriate indexes that already maintain the correct ordering. This is exactly the case in our scenario where *eId* (i.e. the primary key of T) implicitly comprises an index.

```
DELETE /*+ PARALLEL */ FROM W
WHERE eId IN (
   SELECT eId FROM (
      SELECT eId FROM W
      WHERE class = c1 AND ROWNUM <= λ(c1)
      ORDER BY eId ASC
   ) Wc1
   UNION ALL
   ...
   UNION ALL
   SELECT eId FROM (
      SELECT eId FROM W
      WHERE class = ck AND ROWNUM <= λ(ck)
      ORDER BY eId ASC
   ) Wck
)
```

Listing 9.1: Delete statement to rebalance the table-based partial memory

Due to the inclusion of new examples and the removal of obsolete ones from the previous paragraph, pattern statistics need revision accordingly. Before this action can be performed, M has to be synchronized upfront with the new content of W because it essentially manages the rule coverage per example. Thus, we can follow two approaches: (i) only maintain the proportions of data in M that are affected by the population and rebalacing activities of W or (ii) rebuild M from scratch. While the former obviously requires several modifications to reassemble M, the latter can be performed using a simple SQL statement. For the sake of illustration, we decide on (ii) but clearly state that (i) is more attractive in situations where only small pieces of M need to be revised.

Consequently, (ii) is conducted by using a "full outer join" between W and R, which is appended to the emptied table M. The processing is depicted in Listing 9.2 where the join predicate is applied to all condition features based on the $*$-symbol introduced earlier. As a result, M comprises the complete mapping between examples and patterns including both examples that are not covered by any rule and patterns that are not matching a particular example. After performing this initial rectification activity on M, we are able to update the pattern statistics located in R. Hence, we can use a grouping operation on M to count correct matches as well as overall hits for each single decision rule denoted by the attributes *posHitCnt* and *hitCnt*. In a final step, these obtained figures are merged into R concluding the updating process, which is illustrated in Listing 9.3. Note, feature *delta* symbolizing the error of a rule is not part of the update because it is a computed virtual attribute depending on *posHitCnt* and *hitCnt* respectively.

```
INSERT /*+ PARALLEL */ INTO M
SELECT eId, rId, class, decision FROM W
FULL JOIN R ON
    (W.a1 = R.a1 OR R.a1 = '*') AND ... AND (W.am = R.am OR R.am = '*')
```

Listing 9.2: Insert statement to maintain the rule coverage among all examples in the partial memory

```
MERGE /*+ PARALLEL */ INTO R
USING (
    SELECT rId, SUM(posHit) AS posHitCnt, SUM(hit) AS hitCnt FROM (
        SELECT rId,
            CASE
                WHEN class = decision THEN 1
                ELSE 0
            END AS posHit,
            CASE
                WHEN eId IS NULL THEN 0
                ELSE 1
            END AS hit
        FROM M
        WHERE rId IS NOT NULL
    ) GROUP BY rId
) H ON
    (H.rId = R.rId)
WHEN MATCHED THEN
    UPDATE SET R.hitCnt = H.hitCnt, R.posHitCnt = H.posHitCnt
```

Listing 9.3: Merge statement to update rule statistics

So far, we took care of consistency aspects, which are important right after examples are populated to W. In this paragraph, we outline the dropping condition heuristic in order to illustrate the generalization of existing patterns. By analogy to the previous section, we start with randomly sampled rules G, which were not simplified earlier. In this context, Algorithm 9.3 conceptually suggests an iterative process to compute coverage and error of one pattern at a time. This approach can be considered inefficient in relational terms, and we follow a different SQL implementation instead. It has the advantage to perform all of these actions at once and scans W only one time. For this purpose, we first generate all possible rule candidates from G by examining the original rule descriptions and replace literals of the form $a = v$ with $a = *$ for all condition attributes $a \in A$ and $v \in V_a$ that are not already marked as irrelevant literal. As a result

```
FOR a_i IN (SELECT V FROM TABLE(FCT_GETCONFIG('ATT_LIST'))) LOOP
   INSERT /*+ PARALLEL */ INTO GS
   SELECT rId, a_1, ..., '*' AS a_i, ..., a_m, decision FROM G
   WHERE a_i <> '*';
END LOOP;

INSERT /*+ PARALLEL */ INTO CR (rId, a_1, ... a_m, decision, length, posHitCnt, hitCnt)
SELECT GW.rId, GW.a_1, ..., GW.a_m, GW.decision, R.length -1 AS lengthNew,
   GW.posHitCntNew, GW.hitCntNew
FROM (
   SELECT G.rId, G.a_1, ..., G.a_m, G.decision, G.posHitCntNew, G.hitCntNew,
      CASE
         WHEN G.hitCntNew = 0 THEN 0.0
         ELSE 1-(G.posHitCntNew/G.hitCntNew)
      END AS deltaNew
   FROM (
      SELECT G.rId, G.a_1, ..., G.a_m, G.decision,
         SUM(
            CASE
               WHEN G.decision=W.class THEN 1
               ELSE 0
            END
         ) AS posHitCntNew,
         SUM(
            CASE
               WHEN W.eId IS NULL THEN 0
               ELSE 1
            END
         ) AS hitCntNew
      FROM GS G
      LEFT JOIN W ON
         (G.a_1 = W.a_1 OR G.a_1 = '*') AND ... AND (G.a_m = W.a_m OR G.a_m = '*')
      GROUP BY G.rId, G.a_1, ..., G.a_m, G.decision
   ) G
) GW
JOIN R ON
   R.rId = GW.rId
WHERE R.hitCnt < GW.hitCntNew AND GW.deltaNew <= (SELECT CAST(
      V AS DECIMAL(18,4)) FROM TABLE(FCT_GETCONFIG('BETA')));

DELETE /*+ PARALLEL */ (
   SELECT * FROM CR
   JOIN (
      SELECT rId, MIN(delta) AS deltaMin FROM CR
      WHERE length = j - 1
      GROUP BY rId
   ) CRB ON
      CR.rId = CRB.rId
   WHERE CR.length = j - 1 AND CR.delta > CRB.deltaMin
);
```

Listing 9.4: Qualitative extract of the SQL procedure to generalize patterns of equal length using dropping conditions

of this process, the candidate sets are appended to an empty temporary table GS and are ready to be compared with W. This comparison is performed by a "left outer join" seeking for matches, which are leveraged next to compute new statistics of candidates in GS using a grouping operation. Based on that, we obtain the statistics. These are

correlated to corresponding parent rules in R using a conventional join operation. Only those child rules with a higher coverage and an error equal or below β are retained and stored in a temporary table CR ultimately. After these activities, CR can still contain multiple child rules for a single parent rule. This is handled in an additional step to determine those child patterns with lowest error employing a "self join". Only those remain inside CR, while siblings with higher δ are deleted. In this regard, a "user-defined function" is declared, which encapsulates configuration Z in order to extract its values conveniently according to a given key. All of these actions are illustrated in Listing 9.4 qualitatively.

Given these aspects as baseline, it should be noted that our approach can be implemented in most modern relational DB systems such as Oracle DB, PostgreSQL or Microsoft SQL Server. However, each of these systems carries particularities due to syntactical and conceptional deviations, which complicates a unified depiction. To get around this issue, we restricted the practical description in this section to Oracle's SQL dialect detailing related SQL expressions from a qualitative angle. In this respect, we came across several new language constructs including the keywords "INSERT", "MERGE", "ORDER BY" or different "JOIN" variants. Even though their usage and meaning was oftentimes self-explanatory and contextualized to a certain extent, an explicit and detailed description could not be provided due to a rather high technical dimension. For that reason, we relegate the interested reader to relevant literature for the sake of completeness in order to get deeper insights to these language constructs and related concepts of DB systems eventually: [140, 190, 191, 445, 446]. It should be further emphasized that due to the general-purpose realization outlined in this section, several minor modifications need to be considered in terms of InDBR's integration to the overall HFIDS architecture. Consequently, details in this direction are deferred to Chapter 10.

9.4 General Evaluation Under Drifting Conditions

This section outlines a general evaluation of the proposed pattern building algorithm InDBR by contrasting its performance from different angles with other related incremental rule learning approaches. First, we describe the underlying experimental setup with competing algorithms and utilized data sets (Section 9.4.1). Against this background, the predictive capabilities of each single learner are examined secondly (Section 9.4.2). Furthermore, we evaluate discovery-oriented aspects (Section 9.4.3) followed by assessing time demands in the exposed experimental environment (Section 9.4.4).

9.4.1 Experimental Setup

One of the key design aspects of InDBR is to learn incrementally in nonstationary environments, which manifests that a fair comparison can only be obtained with other rule inducers holding equivalent capabilities. Therefore, G-eRules and VFDR[6] are employed

[6] The implementation available for our evaluation implicitly adjusts to drifts and uses an unordered set of rules for the classification task, which, according to [98, 418], yields best performances.

as two top-down rule-based classifiers using default parameters unless better values were documented. Moreover, we tried to get the sources of other state-of-the-art rule learners such as FACIL, FLORA and RILL representing closely related bottom-up approaches or AQ11-PM+WAH and Hoeffding Rules, but, unfortunately, these implementations were unavailable at the time of writing despite all of our effort. Beside related algorithms acting as a reference for InDBR, a general evaluation under drift requires a diverse collection of benchmark data sets reflecting changing conditions. To arrange such a nonstationary setting, we incorporate four synthetic data sets and five real-world data sets in our experimental setup. Seven of these are assembled using the well-known stream mining framework MOA [426] and its designated online library available at the official website[7], while two are downloaded from a GitHub repository[8]. In what follows, we briefly describe the main ideas behind these data sets and highlight parameters we set in order to acquire desired drift characteristics.

Airline: The task comprised within this data set deals with the classification of flight records being delayed or on-schedule. It contains 539383 records with seven condition attributes covering a nonstationary real-world problem [447]. On that account, it is often utilized to evaluate algorithms under drifting conditions (e.g. [412, 418]). In our experiments, we use a version of this very competitive task available at MOA.

Electricity: This data set contains pricing developments from different Australian electricity markets. Originally outlined in [448], it is frequently employed as benchmark for drifting environments (e.g. [427, 449]) as it expresses price dynamics of demand and supply. Data gathering started in May 1996 and ended in December 1998 summarized by 45312 data points. Each of the records refers to a 30 minute time period and embody eight condition features. The underlying classification problem is concerned with the relative price change within the last 24 hours. The majority class holds 58% of the data and, thus, there is a tendency towards a skewed class distribution. A variant of this data source with normalized features is downloaded from the MOA website.

Outdoor-Stream: A collection of image sets recorded by an autonomous system in a garden environment is encoded by this data set. First used in [450], each of the 4000 records consists of 21 condition features that represent ten images depicting obstacles from different perspectives under different lighting conditions in temporal order. The classification task deals with the separation of each record into 40 different categories. These classes are evenly distributed and certainly encompasses a real-word problem. It is downloaded from the mentioned GitHub repository.

Poker-Hand: Given five playing cards out of a 52-card deck, the challenge of this data set is to predict the correct poker hand. The problem comprises 829201 hands and each is encapsulated by suit and rank resulting in ten condition features. The proportion of the eight smallest classes encloses not more than 7% constituting a highly imbalanced class distribution. Thus, the data set is quite competitive and commonly used in nonstationary environments (e.g. [427, 449]). We employ a normalized version of it available at the MOA website.

[7] `https://moa.cms.waikato.ac.nz/datasets/`

[8] `https://github.com/vlosing/driftDatasets/tree/master/realWorld/` (commit: 89f1665ed89af78caecabec62c680a57a4f16646)

Radial Basis Function (RBF): As part of MOA, this generator is utilized to build a data set with gradual drift characteristics. It randomly positions a fixed number of centroids in the data space with associated class labels and weights. According to their label and weight, new examples are generated and normally distributed around the centers. After the data generation process completes, drift is introduced by moving the centroids in the data space, which is controlled by velocity parameter v. In our setup, we consider 100000 examples, ten features and $v = 0.001$ for two centroids. Note that the class distribution is skewed where 17.60% of the data belong to the minority class.

Rotating Hyperplane (RHP): This data generator presented in [451] can be described as follows: given a d-dimensional space of uniformly distributed data points x, a hyperplane $\sum_{i=1}^{d} w_i x_i = w_0$ is employed to separate the data space. In this respect, points satisfying $\sum_{i=1}^{d} w_i x_i \geq w_0$ are assigned to the positive class or otherwise associated with the negative class where x_i refers to the i-th coordinate of x and w_i is a corresponding weight. A nonstationary environment can be established by changing the orientation and position of the hyperplane altering w_i with the probability of changing direction τ and magnitude c per x, i.e. $w_i = w_i + c\tau$. We use MOA to generate two data sets employing the parameters $\tau = 0.03, c = 0.1$ and $\tau = 0.01, c = 0.1$ to obtain a long lasting and a shorter gradual drift over 200000 points with ten condition features neglecting noise. Note, the former data set also contains notions of local abrupt drifts.

SEA-Concepts: Introduced in [376], this data generator permits to model sudden drift behaviors with three condition attributes whereas only two are relevant for the classification. The decision boundary is computed via different functions based on the two relevant features over time. Using MOA, a data set is created with three classification functions using the first 100000 records while ignoring noise, which results in a binary classification problem. The majority class comprises around 76% of the data. Note, this generator is frequently used in the context of nonstationary environments (e.g. [411, 412]).

Weather: This data sets, proposed in [365], is concerned with weather forecasting. Therefore, 18159 data points are provided describing eight meteorological features. These were gathered between 1949 and 1999 at an Air Force Base in North America. Due to the long-term data capturing process, the authors claim that realistic precipitation drift problem is comprised. The distribution of the two classes, explaining whether it is raining or not, is imbalanced where roughly 69% of the ground truth indicates no raining conditions. The data sets is downloaded from the same location as Outdoor-Stream.

These details given, further attention has to be paid to stratify a unified setup among the three rule learning algorithms in two directions. On the one hand, most of the benchmark data sets comprise a mixture of numeric and nominal data types, which refused a direct application of InDBR. Recalling that its current version is only capable to handle categorical data (see Section 9.3.1), a discretization of continuous attributes is inevitable. Approaching this way has no impact to G-eRules and VFDR as they can manage both kinds of data, and so we carefully discretize relevant benchmark data preserving original drift characteristics. A summary of all assembled data sets is depicted in Table 9.1. On the other hand, a reconsideration of the operating mode is an obligatory step in addition. To harmonize the processing of all three learners, we enforce InDBR

Data set	# of rec.	# of atts.	# of class.	Imb.	Type of drift
Airline	539383	7	2	no	unknown
Electricity	45312	8	2	(yes)	unknown
Outdoor-Stream	4000	21	40	no	unknown
Poker-Hand	829201	10	10	yes	unknown
RBF	100000	10	2	yes	gradual
RHP-Long	200000	10	2	no	sudden/gradual
RHP-Short	200000	10	2	no	gradual
SEA-Concepts	100000	3	2	yes	sudden
Weather	18159	8	2	yes	unknown

TABLE 9.1: Summary of general benchmark data sets used in this evaluation showcasing the number of records (rec.), the number of conditions (atts.), the number of class labels (class.), an indicator for data imbalance (imb.) and their inherent type of drift

to treat examples sequentially, which is stipulated by G-eRules and VFDR supporting example-by-example processing only. As a result, we finally obtain a unified setting for a qualitative evaluation along predictive and descriptive capabilities. Note, special care needs to be taken when considering the runtime behavior of the algorithms, which is revisited in the respective section.

9.4.2 Predictive Capabilities

Reviewing Section 9.3.2, class imbalance can be crucial for a ML algorithm particularly in the context of concept drift. At the same time, this phenomenon also causes trouble when metering the predictive capabilities of a learned hypothesis because popular measures such as ACC (see (7.3)), highlighting the ratio between correctly classified cases and all examples seen during the evaluation, can be misleading. In order to showcase its intrinsic problem, let us consider a binary classification scenario where the class distribution is skewed and 970 of 1000 test cases belong to the negative class. A naive classifier having a sense of the class distribution can easily reach $ACC = 97\%$ by assigning each of the 1000 examples to the negative class. Obviously, this classification outcome looks auspicious at first sight but suppresses the fact that the learner produces an error of 100% on positive samples. Hence, it is imperative to select a more sophisticated performance measure to evaluate the predictive capabilities of G-eRules, VFDR and InDBR for both balanced and imbalanced classification problems with two or more classes (see Table 9.1). In order to compensate such situations in our experimental setup, we utilize the "$F1$-score" combined with two established scaling methods, i.e. "micro-averaging" and "macro-averaging". It reflects the harmonic mean of the two measures "precision" and "recall"[9] typically employed in information retrieval, which is a field highly subjected to class imbalance (e.g. [452, 453]). In this regards, the micro-averaged $F1$-score ($\mu F1$) is computed by

$$\mu F1 = 2 \cdot \frac{P_\mu \cdot R_\mu}{P_\mu + R_\mu} \tag{9.9}$$

[9] Recall is another name for hit rate or TPR, which we already introduced through (7.1).

where micro-averaged precision (P_μ) and recall (R_μ) are given through

$$P_\mu = \frac{\sum_{j=1}^{k} TP_j}{\sum_{j=1}^{k} TP_j + FP_j} \text{ and } R_\mu = \frac{\sum_{j=1}^{k} TP_j}{\sum_{j=1}^{k} TP_j + FN_j} \tag{9.10}$$

with TP_j, FN_j and FP_j are the TP, FN and FP for each associated class $c_j, 1 \leq j \leq k \in \mathbb{N}$. This measure equally weights each decision performed by the learner and thus corresponds to the conventional ACC. On the contrary, the macro-averaged $F1$-score ($mF1$) weights predictions per class evenly facilitating to obtain insights to the effectiveness of a learner across classes. Therefore, it is utilized as additional measure in our experiments. Formally, it is calculated by

$$mF1 = 2 \cdot \frac{P_m \cdot R_m}{P_m + R_m} \tag{9.11}$$

with the macro-averaged precision (P_m) and recall (R_m):

$$P_m = \frac{1}{k} \cdot \sum_{j=1}^{k} \frac{TP_j}{TP_j + FP_j} \text{ and } R_m = \frac{1}{k} \cdot \sum_{j=1}^{k} \frac{TP_j}{TP_j + FN_j} \,. \tag{9.12}$$

As such, we obtain two performance measures rating the overall classification in terms of correct predictions and a classifier's deficits on data imbalance.

A further challenge in our experimental setup is the incremental nature of the algorithms in charge. While conventional evaluation methods typically rely either on hold-out test sets or cross-validation, the model of an incremental learner evolves over time just as the data. These circumstances refuse an immediate utilization of the introduced $F1$-scores $\mu F1$ and $mF1$ because no substantial test data are disposable given the underlying continuous learning cycle especially under drift. To counter these issues, a common methodology capable to measure the predictive performance is the "predictive sequential" (prequential) or "interleaved test-then-train" technique (see [369, 454, 455]). In essence, it first utilizes the existing model to predict the class of an incoming example right before the example with its true label is leveraged to update the model. As such, the model is always tested on examples it has not seen before. This simple idea incorporates both training and testing at the same time and is, therefore, applied in our setup eventually. Combined with a sliding window measuring the $F1$-scores on micro-scale and macro-scale, an additional forgetting strategy is employed not favoring periods containing high errors during drifts or long-lasting intervals where a model is stable constituting pessimistic error estimates.

Using the prequential approach and $F1$-scores basically enable us to evaluate all three rule-based learners. However, special attention has to be paid for two distinguishing concepts pursued by considered algorithms limiting a direct comparison, i.e. the abstaining characteristic and the any-time property (see Section 9.2 and 9.3.3). While G-eRules and InDBR explicitly refrain from classification when they are uncertain about a decision, VFDR is able to provide a prediction at any time even though the rule set is empty or an appropriate rule match is missing. To obtain this capability, VFDR incorporates an enhanced default rule that applies conventional NB functionality, which can be

computed straightforwardly using gathered statistics stored in the rule consequent. As argued earlier, such techniques are not contributing much to the transparency objectives of rule learning for critical applications since they constitute black box classification in general. Hence, detaching the default rule from VFDR seems reasonable to unveil the true potentials of its rule engine towards interpretable decision-making, which indeed disclose abstaining characteristics. Following this perception, we meter the any-time capabilities of VFDR alongside with its abstaining behavior. By definition, abstaining is directly comparable with the other two rule inducers whereas the the any-time performance is not. Thus, we temporarily rework InDBR with any-time abilities by training a separate NB model in parallel. This supplementary model relies on the last 1000 examples seen over the course of learning and is intended to take over classification when InDBR is uncertain. Proceeding this way is quite similar to the approach taken by VFDR, which yields comparable $F1$-scores among the two algorithms. Considering the isolated assessment of G-eRules and InDBR, corresponding "abstain rate" (ABR) as well as "tentative $\mu F1$" (t-$\mu F1$) and "tentative $mF1$" (t-$mF1$) are presented. In this context, ABR reflects the percentage of unanswered classification requests due to uncertainty whereas t-$\mu F1$ and t-$mF1$ refer to the predictive capabilities of G-eRules and InDBR in those cases they are confident about a decision.

Given these preliminaries, we turn to the predictive performance analysis of VFDR and InDBR in terms of $\mu F1$, $mF1$ and their abstaining characteristics. Considering the pairwise comparison of $\mu F1$ reveals fairly kindred results not exceeding a maximum difference of 3.70% except for the two data sets Outdoor-Stream and Poker-Hand. On these two benchmark data sets, InDBR performs better by more than 27.90% on the former and more than 6.80% on the latter. A similar behavior can be observed for $mF1$ where in seven out of nine data sets the maximum difference is below 3.80%. Only on RBF, VFDR outruns InDBR by a difference of 11.98%. Yet, VFDR produces more misclassifications across classes on Outdoor-Stream, which is a 20.30% lower $mF1$ compared to InDBR. These findings are confirmed by a low averaged pairwise difference for both $F1$-scores ($< 1.93\%$) when neglecting the mentioned exceptions indicating comparable performances assessing the two algorithms in an any-time setting. However, the abstaining characteristics of both rule engines disclose different insights. In eight out of nine cases, our proposed approach outperforms VFDR with an abstaining that is at least 19.50% lower on average. VFDR only can compete with InDBR on Poker-Hand in this category. On this particular data set, it reaches an averaged ABR of 32.37% over the course of the learning, which constitutes roughly 2.60% more abstentions compared to InDBR. In turn, the maximum difference between both algorithm is obtained on Airline where InDBR only abstains in 13.59% of the cases whereas the ABR of VFDR reaches 94.63% demonstrating an average difference of 81.04%. Overall, the averaged refraining behavior of InDBR accounts for 16.30% on all nine data sets. This completely contrasts to the ABR of VFDR carrying a mean value of 60.46%. From a rule learning perspective, the poorest result of VFDR is conducted on Outdoor-Stream not supplying any rule match, i.e. $ABR = 100.00\%$. This circumstance, on the one hand, explains the fragile predictive performance because a simple NB model learned on this data set performs almost arbitrarily and so does VFDR. On the other hand, this outcome literally signifies its general drawback, i.e. a lazy rule induction especially in the initial learning phase with few training examples available. This key observation is further stressed by

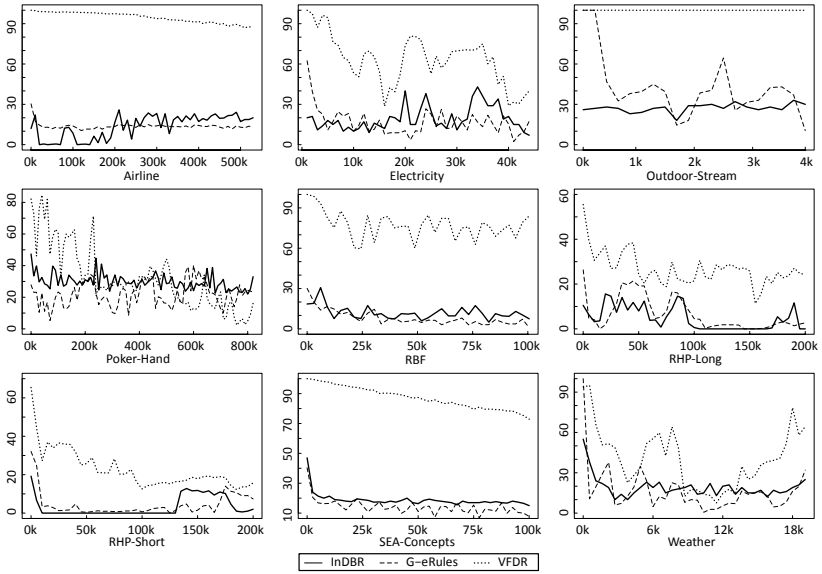

FIGURE 9.4: *ABR* of all rule inducers in percent over the course of the incremental learning process on all benchmark data sets

Figure 9.4 detailing VFDR's abstaining characteristics over the complete learning cycle on all nine benchmark data sets in comparison to InDBR and G-eRules.

The results assessing InDBR and its direct contender G-eRules are two-fold. Even though both pure rule engines share nearly equivalent outcomes on a pairwise comparison for t-$\mu F1$ with an average difference of 3.12% on five data sets with a maximum discrepancy of 4.60%, a much weaker performance is observed for G-eRules on the other four benchmark tests with a minimum and mean difference of 8.34% and 18.20% respectively compared to InDBR. These results are supported by t-$mF1$ where InDBR clearly outcompetes G-eRules by one more data set with a deviation of 18.48% on average worth noting that three of these five classification tasks comprise class imbalance. Concerning all five imbalanced benchmark data sets, InDBR also reveals better results with an averaged t-$mF1$ of 74.68%, which is roughly 9.70% higher than the figures produced by G-eRules. Its potentials become even more convincing when reviewing the performance on multiclass problems represented by Poker-Hand and Outdoor-Stream in our series of experiments. An averaged t-$mF1$ of 82.83% produced by InDBR faces a 33.52% better result compared to G-eRules. As opposed to these bare numbers, further details of the performance characteristics are presented in Figure 9.5. It graphically illustrates the prequential performance measured by t-$mF1$ over the entire learning process of both algorithm on all benchmark tests. This perspective explicitly highlights the benefits of InDBR where both rule engines only show comparable numbers on RBF and both RHP variants. Particularly on the RHP data sets, we can see an overlapping behavior for some ranges between the two learners, but at the same time segments in the graphs

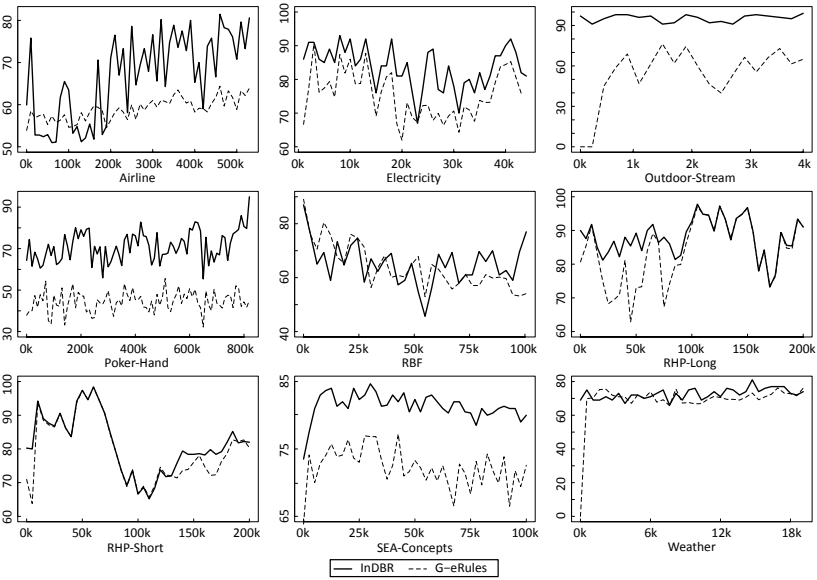

FIGURE 9.5: Prequential performance of G-eRules and InDBR in percent using t-$mF1$ over the course of the incremental learning process on all benchmark data sets

can be identified where the classification abilities of G-eRules collapses and InDBR's performance remains stable. On all other benchmark data sets, InDBR documents better outcomes for the most part reflecting solid results given the degree of difficulty on these data sets. Conflating these points indicate that our rule learning approach in combination with the partial memory is promising and provides a broader visibility over the course of the learning cycle. Turning over to *ABR*, both rule inducers demonstrate no compelling disparity on seven benchmark tests with an averaged difference of 1.46% on a pairwise comparison. This assessment is confirmed reviewing Figure 9.4 where only a notable distinctions can be observed on two data sets. While G-eRules has 7.91% fewer abstentions on Poker-Hand, InDBR conducts 15.02% more predictions on Outdoor-Stream equalizing the two protruding cases. With these abstaining behaviors at hand, an intuitive tie can be argued for both rule learning algorithms in the given experimental setup.

So far, we presented tendencies comparing VFDR, G-eRules and InDBR in the last two paragraphs. Nonetheless, these observations are not providing sufficient evidence to finally rate the predictive performances at least from a statistical point of view. Therefore, the Friedman test [456] and the Wilcoxon signed-ranks test [457] are incorporated as suggested in [458] to verify the previous assessments more formally. While in the remainder of this paragraph we report on obtained results applying both tests, further details about test procedures and related statistics are described in Appendix A.3 and A.4 respectively. Note, all tests are conducted with a standard level of significance $\alpha = 5\%$, which implies that an attested significant difference by either test might be

Data set	VFDR		G-eRules		InDBR			
	$\mu F1$	$mF1$	t-$\mu F1$	t-$mF1$	$\mu F1$	$mF1$	t-$\mu F1$	t-$mF1$
Airline	**66.40**	61.70	62.97	59.00	66.19	**65.46**	**67.57**	**66.85**
Electricity	78.88	78.62	76.24	75.36	**80.27**	**79.61**	**84.59**	**83.61**
Outdoor-Stream	55.43	62.01	57.01	54.13	**83.33**	**82.33**	**95.33**	**94.67**
Poker-Hand	78.66	**63.85**	77.32	44.48	**85.48**	63.45	**94.88**	**70.99**
RBF	**90.04**	**85.26**	83.92	63.98	86.86	73.28	**86.00**	**64.95**
RHP-Long	83.41	83.51	83.71	83.73	**86.84**	**86.85**	**87.97**	**87.97**
RHP-Short	**81.61**	**81.62**	79.64	79.66	80.52	80.56	**81.29**	**81.32**
SEA-Concepts	86.06	**81.58**	81.76	72.22	**86.51**	80.42	**90.31**	**81.47**
Weather	71.48	67.68	76.63	68.85	**75.17**	**70.28**	**79.67**	**72.39**
Mean rank	1.67	1.56	2.00	2.00	**1.33**	**1.44**	**1.00**	**1.00**

TABLE 9.2: Average results of the prequential evaluation in percent using $\mu F1$, $mF1$, t-$\mu F1$, t-$mF1$ including mean ranks; bold numbers indicate overall winner per row and performance measure

potentially wrong with a chance of 5%. Starting with the comparison of the $F1$-scores on VFDR and InDBR, neither test reveals an indication to reject the null hypothesis stating no noticeable distinction between the predictive performances in our experimental setup. Completely different insights are given when comparing the *ABR* of VFDR and InDBR where both tests agree on the alternative hypothesis concluding a substantial discrepancy among the two algorithms in this discipline. The same can be exposed by contrasting the tentative $F1$-scores of G-eRules and InDBR yielding a statistical difference that is significant. The comparison of the abstaining behavior are rated to be nearly identical either using the Friedman or Wilcoxon test. These results entirely confirmed the described tendencies from the previous two paragraphs. However, it is arguable that the weak performances of VFDR and G-eRules on the unrepresentative Outdoor-Stream benchmark bias the hypothesis tests. But even ignoring this data set unveils the same results. Given these solid outcomes acknowledged by two statistical hypothesis tests, we finally can conclude that InDBR's any-time capabilities are comparable to the state-of-the-art rule learner VFDR. This point can even be examined by reviewing their mean ranks, which are fairly close either on $\mu F1$ or on $mF1$ signifying that both algorithm have a balanced win-loss relation in our experimental setup. When it comes to transparent decision-making, InDBR performs better with a much lower abstaining behavior compared to VFDR winning the comparisons on all nine benchmark data sets. Contrasting InDBR with its direct competitor uncovers that its tentative predictive performance significantly outpaces G-eRules at all levels with a mean rank of 1.00 on t-$\mu F1$ and t-$mF1$. Finally, we conclude that their abstaining characteristics behave alike not least because of their narrow mean ranks obtained over the course of our experiments, i.e. a rank of 1.44 for G-eRules and a rank of 1.56 for InDBR. A summary of all reported results is depicted in Table 9.2 and 9.3. Selected parameter settings of InDBR are given through Table 9.4.

In addition to the discussed outcome, it is interesting to conduct further experiments to analyze the prequential behavior of all three rule learning algorithm under noisy circumstances. Therefore, we reuse the existing setup with all nine data sets but add 15% noise to their GT at random in order to receive insights to their predictive performances when training data tend to be imperfect. In this setting, the pairwise comparison of

Data set	VFDR	G-eRules	InDBR
Airline	94.63	13.70	**13.59**
Electricity	62.55	**16.76**	19.39
Outdoor-Stream	100.00	42.02	**27.00**
Poker-Hand	32.37	**21.84**	29.74
RBF	76.85	**8.02**	11.98
RHP-Long	26.69	6.61	**4.83**
RHP-Short	22.80	4.30	**3.26**
SEA-Concepts	87.35	**14.09**	18.68
Weather	40.88	**16.28**	18.22
Mean rank	3.00	**1.44**	1.56

TABLE 9.3: Average *ABR* of the rule learners during the prequential evaluation in percent including mean ranks; bold numbers indicate overall winner per row

Data set	Without noise			With noise			Shared	
	w_κ	β	t	w_κ	β	t	α_c	$g(j)$
Airline	750	35.00	6	750	35.00	7	1250	$-0.150j + 1.190$
Electricity	500	20.00	5	500	25.00	5	1000	$-0.130j + 1.140$
Outdoor-Stream	50	0.00	3	50	30.00	4	100	$-0.040j + 0.960$
Poker-Hand	500	0.00	4	500	20.00	5	1000	$-0.096j + 1.073$
RBF	1000	25.00	3	750	27.50	3	1500	$-0.096j + 1.073$
RHP-Long	250	10.00	3	250	20.00	4	750	$-0.096j + 1.073$
RHP-Short	500	18.00	5	500	20.00	5	2000	$-0.096j + 1.073$
SEA-Concepts	500	30.00	4	1000	35.00	4	1500	$-0.770j + 2.420$
Weather	1250	15.00	3	1250	20.00	3	1500	$-0.130j + 1.140$

TABLE 9.4: Parameter settings of InDBR used during the experiments; α_c and $g(j)$ are identical under noise and in the noise-free setup; β-values are presented in percent

VFDR and InDBR reveals no essential different tendencies in contrast to the noise-free setup. Only on Poker-Hand, a discrepancy is observable for InDBR with a decreased $\mu F1$ of 4.78% and 8.56% for $mF1$ whereas VFDR remains stable. Despite this finding, InDBR is still able to win all confrontations measured by $\mu F1$ and manages to win six comparisons on $mF1$. To judge the overall performance under noise, we inspect the average $F1$-scores with and without noise. Results show that their predictive capabilities deteriorate by not more than 2.60% as opposed to the original setup unveiling solid result. In terms of refraining characteristics, the outcome clearly states that InDBR's *ABR* increases on all data sets with the exception of RHP-Long and SEA-Concepts where results change only marginal compared to the noise-free setting. This contrasts with the abstaining of VFDR where on seven benchmark tests the rate can be maintained or even improved. However, refraining increases drastically on data set Weather by more than 23.85%. Combined, this leads to 5.78% more abstaining on average for InDBR whereas VFDR's behavior increases by only 1.85% in contrast to scenarios with no class noise. The assessment of G-eRules and InDBR is discussed next where the recordings of t-$\mu F1$ show a spreading dominance for InDBR on three more data sets such that the minimum and mean deviation to G-eRules amounts for 8.16% and 16.01% on seven benchmark sets. Only on RHP-Long and RHP-Short, both algorithms are comparable using t-$\mu F1$. When it comes to contrasting those rule learners applying t-$mF1$, similar results can be discovered except for Airline and RBF. On these two data sets, InDBR still wins the prequential evaluation but margins are not as big as utilizing t-$\mu F1$. Yet, the performance gap to G-eRules raises on Weather from 3.54% to 8.35% such that

Data set	VFDR		G-eRules		InDBR			
	$\mu F1$	$mF1$	t-$\mu F1$	t-$mF1$	$\mu F1$	$mF1$	t-$\mu F1$	t-$mF1$
Airline	64.31	**60.10**	57.95	55.10	**66.15**	58.10	**68.55**	**59.01**
Electricity	76.88	76.02	69.70	69.38	**79.58**	**78.98**	**84.82**	**83.67**
Outdoor-Stream	51.00	56.53	50.27	48.14	**75.00**	**82.00**	**86.00**	**89.00**
Poker-Hand	79.93	**62.59**	67.95	43.66	**80.70**	54.88	**90.58**	**62.71**
RBF	85.70	**81.63**	77.40	62.27	**86.29**	72.87	**85.55**	**65.48**
RHP-Long	83.34	83.44	83.05	83.11	**86.30**	**86.31**	**87.63**	**87.63**
RHP-Short	81.35	81.40	82.12	82.23	**81.87**	**81.87**	**83.82**	**83.81**
SEA-Concepts	80.36	73.72	77.77	70.96	**83.98**	**76.54**	**88.70**	**78.73**
Weather	71.21	67.08	69.39	64.07	**72.94**	**69.11**	**78.31**	**72.42**
Mean rank	2.00	1.67	2.00	2.00	**1.00**	**1.33**	**1.00**	**1.00**

TABLE 9.5: Average results of the prequential evaluation in percent with 15% class noise using $\mu F1$, $mF1$, t-$\mu F1$, t-$mF1$ including mean ranks; bold numbers indicate overall winner per row and performance measure

InDBR outruns its opponent on four out of five imbalanced data sets considerably with a minimum and mean difference of 7.77% and 12.36%. In summary, the discrepancy between G-eRules and InDBR increases by 3.33% on t-$\mu F1$ while it remains the same on t-$mF1$ cumulated over all nine test runs. This illustrates a noticeable win on both t-$\mu F1$ and t-$mF1$ for InDBR overall. Despite the level of class noise, a further interesting observation is InDBR's ability to maintain fairly the same tentative $F1$-scores ($\pm 2.80\%$) without substantial loss in both experimental setups, which is documented on four tasks with class imbalance and additional two balanced data sets. In terms of refraining characteristics, G-eRules and InDBR show no distinct result on a pairwise comparison. While in the noise-free setting both learners perform equally with the exception of two data sets, a higher diversity of wins and losses can be identified. On the one hand, the abstaining of InDBR raises by more than 8.36% on Airline and Poker-Hand. On the other hand, the original gap between InDBR and G-eRules on SEA-Concepts and Weather, catering for 3.27% on average, is minimized with a mean disparity of 0.24% in noisy scenarios. This gap is even expanded on both RHP variants such that G-eRules refrains 4.88% more often. Nevertheless, both algorithm share nearly the same mean abstaining in 21.95% of the cases, which finally leads to a tie under class noise in this category. This is an increase of roughly 5.83% as opposed to the noise-free setup. Considering the application of the Friedman and Wilcoxon tests, most described results are congruent from a statistical perspective using $\alpha = 5\%$. InDBR is significantly better than G-eRules on both tentative $F1$-scores, while there is no remarkable difference on their abstaining behavior. Visually, this is supported by the mean ranks of 1.00 for InDBR and 2.00 for G-eRules on t-$\mu F1$ and t-$mF1$ whereas their ranks are very close in the category of refraining. Deducing a conclusion on *ABR* of VFDR and InDBR, a deviation is notable such that the null hypothesis of both tests has to be rejected despite the fact that the difference between InDBR and VFDR slightly decreased under noise to a mean rank of 2.89. Additionally, there is no statistical discrepancy using $mF1$ just as in the original setting. However, both hypothesis tests indicate a significant difference for $\mu F1$. This constitutes a conflict to our earlier intuitive assessment but can be justified quantitatively given the nine wins of InDBR in this discipline. The results of this setup under noise are highlighted in Table 9.5 and 9.6 respectively whereas chosen parameters are provided in Table 9.4.

Data set	VFDR	G-eRules	InDBR
Airline	95.73	**16.25**	21.95
Electricity	57.30	**22.71**	26.51
Outdoor-Stream	100.00	49.51	**33.00**
Poker-Hand	27.62	**20.80**	39.60
RBF	70.91	**16.58**	19.15
RHP-Long	28.49	14.51	**7.10**
RHP-Short	29.04	15.04	**9.88**
SEA-Concepts	86.92	**16.72**	17.81
Weather	64.73	24.37	**23.75**
Mean rank	2.89	**1.44**	1.67

TABLE 9.6: Average *ABR* of the rule learners during the prequential evaluation in percent with 15% class noise and mean ranks; bold numbers indicate overall winner per row

9.4.3 Discovery-oriented Capabilities

With a focus on transparent decision-making in this chapter, other criteria than predictive capabilities are of certain interest to evaluate a produced rule-based classification model. Particularly in our case where we intend to build interpretable decision rules from arriving examples, a further aspect is concerned with the quality assessment of induced patterns in order to characterize their abilities from a descriptive perspective (see [459, 460, 461]). These include their interestingness or usefulness. Yet, the quantification of such attributes can turn out to be highly subjective and the availability of established metrics varies widely. To overcome this issue, we, therefore, pursue a pragmatical attempt by introducing five distinct measures with a focus on discovery-oriented aspects for nonstationary environments. These are given as follows:

- **Average rule set size:** An indicator for the complexity of a rule-based model is the cardinality of its rule set similar to the size of a tree in DT learning. A small value is favorable with respect to both computational demands and monitoring aspects.
- **Average rule length:** The number of literals of a pattern's condition part provides insights to its level of generality and its simplicity. A shorter rule length tends to cover more cases and is more comprehensible. Hence, there is a broad consensus that a short rule length is preferable. However, longer patterns are closer to the data and can be valuable in addition as they supply greater detail.
- **Avgerage multi-match rate:** Occasionally, several patterns are covering an arriving example simultaneously which we refer to as "multi-matching". From a transparency perspective, such situations can be critical because multiple explanations for a given case are cumbersome for both the rule engine and human interpretability particularly when they are conflicting. Thus, a low multi-match rate is desirable in general.
- **Average coverage:** The coverage of a rule set is essential. It signals how well arriving examples can be modeled by the pattern induction process. Particularly in nonstationary environments, a low coverage points out an adapting deficit and a poor representation of most recent data.

- **Average rule purity:** Besides the complete coverage aspect, the purity of an individual pattern is of interest. It delivers an intuition about the pattern's consistency and confidence for decision-making as a high value indicates greater predictive quality.

In what follows, we apply these five described measures to the rule learning algorithms VFDR, G-eRules and InDBR using the nine benchmark data sets of Section 9.4.1 neglecting class noise. In this regards, the metering of the average rule set size, average rule length and multi-match rate occurs periodically over the course of the learning process. To determine the average coverage and average rule purity, a collection of baseline examples is required. Therefore, we leverage a reference sliding window with a fixed size of the 1000 most recent examples. Quite naturally, the average rule set size and average rule length constitute absolute values, while the multi-match rate, average coverage and average rule purity reflect relative numbers. Note that, in line with the previous examination, we only evaluate the active rule set of InDBR and ignore the default rule of VFDR.

Analyzing the results unveils that VFDR outperforms InDBR in terms of the rule set size on eight out of nine benchmark data sets. On a pairwise comparison, VFDR produces 100 fewer rules compared to InDBR on average. Only on Airline, the difference is in single digits. These gaps between the two learners seem remarkably high but can be explained by the very high ABR produced by VFDR which we uncovered earlier. Another finding in our experiments is the huge average rule set size generated by G-eRules. Except for Weather, the size is at least two times greater in contrast to InDBR. Particularly on three data sets, the disparity is significant. The model complexity of G-eRules is approximately one order of magnitude higher when applied to SEA-Concepts and Poker-Hand. On average, it is even 37 times larger compared to InDBR with respect to the Airline benchmark. These results are exemplified in the first row of Figure 9.6 for Poker-Hand revealing a nearly constant growth of the rule set up to a mean size of 3127.68 whereas numbers of InDBR and VFDR are steadily below 270.00. The descriptive performance of G-eRules turns out to be much more competitive considering the average rule length. On four data sets, differences to InDBR are below 1.64 for both learners signifying equivalent results virtually. On the other five benchmark sets, G-eRules wins three times and looses two times. The lowest numbers in this category by far are produced using VFDR catering eight out of nine wins overall. We relate these results to the different rule induction strategies implying a general disadvantage for InDBR, which relies on bottom-up rule learning as opposed to the top-down approaches of its contenders. The effect behind these contrasting concepts is visualized in the second row of Figure 9.6 where, unlike the other two rule engines, InDBR constantly tries to simplify patterns. From a multi-match standpoint, VFDR and InDBR behave similarly on five data sets with an average difference of 1.51%. This trend is, however, not expanded on the other four benchmark tests. Applying VFDR on these produces 35.56% more multi-matches compared to InDBR despite its relative small average rule set size. G-eRules demonstrates very high numbers in this discipline such that a multi-match scenario can be identified for almost every second classification when averaging the results on all nine benchmark data sets. Putting this into perspective with the other two learners, these rule conflicts only occur in 17.19% of the cases for VFDR and

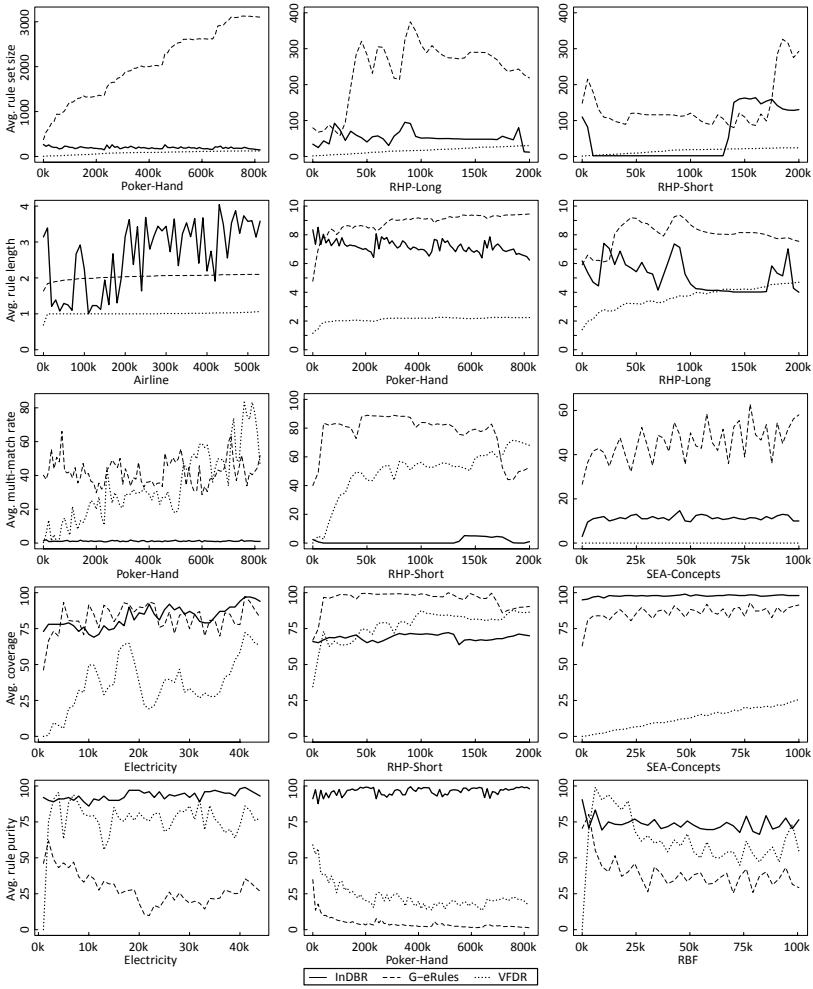

FIGURE 9.6: Discovery-oriented aspects over the course of the incremental learning process on selected data sets; average multi-match rate, coverage and rule purity are expressed in percent

InDBR has a rate of 2.23% on average. This signals a distinctive win for InDBR in our experimental setup. The multi-match results on all algorithm are exemplified by three representative graphs depicted in the third row of Figure 9.6 where the solid low outcome of InDBR is noteworthy. Moreover, results on the rule set coverage show that InDBR outnumbers VFDR on seven data sets with an average of 87.65%. VFDR can only cover 28.99% of most recent examples in this respect. Applying both algorithms to the other two benchmark tests indicates a better result for VFDR where its coverage is 7.69% higher on average. Despite these two losses, InDBR's performance is dominant and

Data set	VFDR			G-eRules			InDBR		
	size	len	mult	size	len	mult	size	len	mult
Airline	92.67	**1.01**	**0.24**	3100.85	2.03	47.51	**83.73**	2.65	3.11
Electricity	**8.81**	**1.06**	**3.06**	896.28	2.80	50.90	114.97	5.72	3.48
Outdoor-Stream	**0.00**	**0.00**	**0.00**	95.90	4.76	8.38	24.90	18.54	**0.00**
Poker-Hand	**82.06**	**2.12**	33.43	2052.38	8.84	41.77	194.65	7.20	**1.15**
RBF	**13.63**	**1.46**	**3.59**	451.70	1.88	66.47	121.75	6.26	4.09
RHP-Long	**16.77**	**3.64**	40.59	242.74	8.03	73.56	52.74	5.15	**0.87**
RHP-Short	**15.73**	3.33	49.04	133.97	8.15	76.53	52.14	**3.19**	**1.10**
SEA-Concepts	**15.83**	**1.01**	**0.00**	2571.57	1.90	45.54	264.60	1.40	3.78
Weather	**8.31**	**1.29**	24.78	369.05	4.47	61.52	252.51	5.82	**2.50**
Mean rank	**1.11**	**1.11**	**1.50**	3.00	2.44	3.00	1.89	2.44	**1.50**

TABLE 9.7: Discovery-oriented aspects of rule learners with respect to average rule set size (size), average rule length (len), average multi-match rate (mult) including mean rank; bold numbers indicate overall winner per row and performance measure; mult is expressed in percent

reaches a mean coverage of 83.19% on all data sets, which is 43.92% higher compared to VFDR. Considering G-eRules and InDBR, results are not that obvious. Both coverages behave equally on four benchmark sets, while on the other data InDBR accounts for three wins and two losses. Three examples of the average coverage characteristic are visualized in the fourth row of Figure 9.6. In terms of rule precision, our experiments reveal that the individual rule quality of InDBR is well ahead of its competitors in at least five out of nine benchmark tests. On these five data sets, InDBR reaches a 47.18% higher purity compared to VFDR. On the other data sets, both rule learners perform equally well except for SEA-Concepts where VFDR performs convincingly better. The comparison of G-eRules and InDBR is even clearer concerning the average purity. On all nine benchmark data sets, InDBR outcompetes G-eRules significantly such that its rules have a 44.13% better precision on average. To illustrate these results, the fifth row of Figure 9.6 outlines the purity on all three learners on three benchmark sets over the course of the learning cycle. Examining the win-loss analysis of our experimental setup for InDBR unveils one win and eight times a second place regarding the average rule set size. This constitutes a mean rank of 1.89, which is behind the outcome of VFDR. Concerning its rule length, InDBR has a mean rank of 2.44, which is the weakest result of all descriptive measures. Yet, it shares this rank with G-eRules indicating similar potentials among the two learners. VFDR clearly dominates this comparison. By means of multi-matching, InDBR wins four times, accounts for one tie and is four times in second position. As a result, a mean rank of 1.50 is obtained that is shared with VFDR. Note, the mentioned tie is received on Outdoor-Stream where VFDR is abstaining entirely, which in turn signals that no potential rule conflict can occur at all as no rule is utilized to make a decision. Taking this observation into perspective denotes a win for InDBR in this category. In terms of the average coverage, InDBR acquires five wins and two times the second and third place with a mean rank of 1.67. This result is very close to the average coverage produced by G-eRules constituting an equal behavior. VFDR is far off and cannot compete in this discipline. Considering the rule purity, InDBR caters for eight wins and is one time in the second position. This yields a solid win highlighted by its mean rank of 1.11. All of these discussed results are illustrated in Table 9.7 and 9.8 accordingly.

Data set	VFDR		G-eRules		InDBR	
	cov	pur	cov	pur	cov	pur
Airline	5.37	19.64	83.39	17.13	**96.38**	**74.15**
Electricity	36.45	76.22	82.10	28.69	**83.00**	**93.23**
Outdoor-Stream	0.00	0.00	**74.26**	67.14	71.40	**80.15**
Poker-Hand	67.21	22.93	78.01	4.33	**89.95**	**96.32**
RBF	22.47	62.54	92.61	39.37	**97.25**	**73.36**
RHP-Long	73.31	87.69	**93.06**	62.40	66.36	**88.10**
RHP-Short	77.21	80.82	**95.52**	59.53	68.78	**82.01**
SEA-Concepts	12.63	**96.55**	85.99	11.90	**97.79**	84.63
Weather	58.79	75.24	**82.77**	64.28	77.81	**79.97**
Mean rank	2.78	2.00	**1.56**	2.89	1.67	**1.11**

TABLE 9.8: Discovery-oriented aspects of rule learners with respect to average coverage (cov) and average rule purity (pur) in percent including mean rank; bold numbers indicate overall winner per row and performance measure

9.4.4 Time Consumption

In this section, we make an attempt to highlight the performance of all three rule learning algorithms in terms of their practical time consumption. Therefore, we reuse our experimental setup hosted on a prepared machine[10] inside a dedicated virtualization platform[11]. This setting is exploited to meter the elapse time for each data set from the beginning of the training until all examples are processed successfully. These measurements are repeated ten times in order to obtain reliable runtimes. As such, we get an intuition of the algorithmic time demands per method applying all nine data sets. However, approaching this way can be considered critical. While this empirical time assessment is fair for VFDR and G-eRules, it conceals the true potentials of InDBR. The main reason for this issue relies on different design philosophies. Recalling that InDBR is designed for in-DB applications that generally perform best when operating on sets of input data, the conceptual perception of VFDR and G-eRules pursues example-by-example processing exclusively. These circumstances make an comparison all the way more difficult with respect to an unbiased time analysis. Despite this dilemma, we continue to present the time consumption of the algorithms from a sequential perspective in order to be compliant with the test configuration utilized in the previous two sections but clearly emphasize that results herein must be interpreted with caution as InDBR can perform better.

Being aware of this matter, let us briefly revisit the Oracle implementation of Section 9.3.4, which is leveraged in the remainder of this section to showcase the sequential processing performance of InDBR. This realization is widely dominated by comparison operations to manage the pattern building process throughout the learning cycle and in turn supported by hash joins and hash aggregations respectively. Already experienced in Chapter 4, these hash algorithms are efficiently implemented by DB systems and have the capabilities to produce highly parallel execution plans as indicated by the query hint "PARALLEL" in the corresponding listings of Section 9.3.4. Consequently, we

[10] Microsoft Windows Server 6.3 (std., build 9600), Oracle DB 12c (enterprise, 12.1.0.2), JDK 1.8.0.51, 8× vCPUs, 16 Gbyte RAM, 256 Gbyte HDD

[11] VMWare ESXi 6.5.0 (build 4564106) on a Fujitsu Primergy RX200 S7: 2× Intel Xeon CPU E5-2630 (2.30 GHz), 96 Gbyte RAM, 4× 1 Tbyte SAS HDD configured as RAID-5

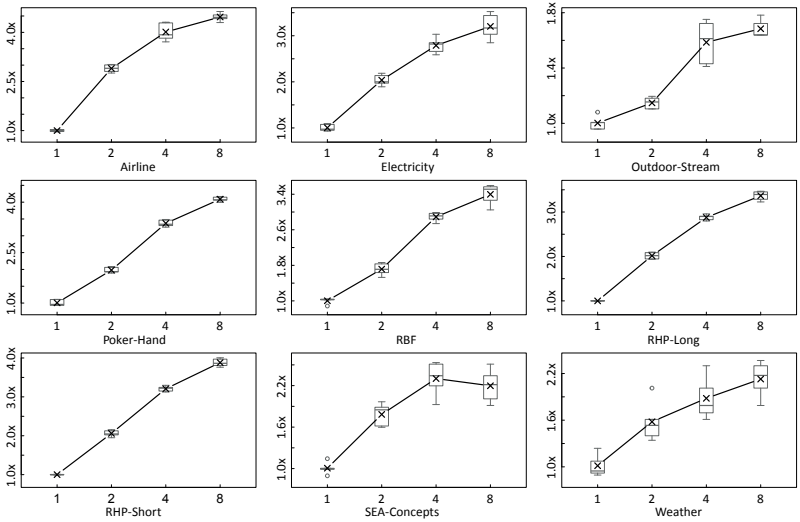

FIGURE 9.7: Speedup of InDBR when processing sequentially given as boxplots; x-axes show the employed # of vCPUs and ×-lines indicate mean tendencies

are able to examine the scaling potentials of the entire implementation. Therefore, we meter InDBR's elapse times on different vCPU configurations in order to determine its attainable speedup per data set. Obtained results show that InDBR can scale effectively. On six data sets, it accomplishes a speedup of 3.73 on average, which signals that its processing performance can be increased by more than three times compared to the utilization of a single vCPU. On the other three data sets, the speedup reaches an average of 2.04 and a maximum value of 2.30 for SEA-Concepts. It is interesting to note that on the latter data set the best outcome is aquired using four vCPUs only. This is in contrast to the other eight benchmark sets where the maximum number of vCPUs, i.e. eight in our virtual test environment, yields the best runtimes. A summary of InDBR's speedup assessment is visualized in Figure 9.7. It clearly manifests that results fall short of expectation as the level of speedup it produces is moderate in relation to the number of vCPUs available.

In contrast to these findings, the level of speedup for VFDR and G-eRules cannot be demonstrated because available implementations are single-threaded only and, thus, they are incapable to run in parallel. While this scaling deficit is a concern in distributed processing scenarios, their application in our experimental setup is solid using one vCPU only. VFDR's average runtime caters for five wins with a mean rank of 1.44 whereas G-eRules preserves four times the best outcome comprising a rank of 1.78 overall. In this respect, InDBR cannot obtain a single win such that a total rank of 2.78 is reached. Yet, the resulting runtimes of InDBR are fairly close to either one of the other algorithms for the most part. In particular, this finding can be exposed on a pairwise comparison. On the one hand, InDBR can compete with VFDR on the three benchmark sets Outdoor-Stream, SEA-Concepts and Weather producing an averaged difference of

Data set	VFDR	G-eRules	InDBR
Airline	252.02	**40.28**	290.69
Electricity	**9.26**	16.72	22.15
Outdoor-Stream	12.46	**0.59**	15.10
Poker-Hand	**485.91**	723.58	610.88
RBF	**9.93**	26.15	34.01
RHP-Long	**13.29**	264.94	253.10
RHP-Short	**9.14**	215.56	231.81
SEA-Concepts	22.22	**11.94**	30.85
Weather	1.56	**0.61**	3.82
Mean rank	**1.44**	1.78	2.78

TABLE 9.9: Average runtimes of the rule learners in seconds per data set including mean ranks; bold numbers indicate overall winner per row

4.51 seconds. On the other hand, contrasting the runtimes of InDBR and G-eRules yields similar results on Electricity, RBF and Weather with a maximum difference of 7.85 seconds. Applied to Poker-Hand and RHP-Long, InDBR even gets away with two wins by more than 11.84 seconds. All of these reported numbers are extractable from Table 9.9. It highlights the average runtime of each algorithm per data set after ten measurements. Note that cumulated numbers of InDBR represent the outcome of the best vCPU configuration.

Based on these results, we finally can conclude that VFDR and G-eRules dominate the empirical runtime assessment providing a very lightweight realization to build and maintain patterns. In this context, however, a key finding is their inability to use multiple CPUs that support the processing. This limitation has severe practical implications as they can neither scale-up nor scale-out, which justifies that a high throughput caused by a real-world application can cause processing overload resulting in a bottleneck that generally cannot be absorbed by the given design of these rule learning implementations. The other extreme is presented by InDBR where true scaling potentials can be demonstrated. Yet, its speedup is far off from being ideal such that obtained runtimes on a single machine are still lower compared to VFDR or G-eRules. Discussed at the beginning of this section, the main reason for this finding can be attributed to the chosen sequential setting dictated by VFDR and G-eRules. In this operating mode, the internals of InDBR are forced to handle an example at a time creating severe overhead for the DB engine that is usually intended to lift batches of data. This setup is highly inefficient from a set-theoretic perspective and cannot be compensated by available query plans and the level of parallelism utilized.

9.5 Discussion and Summary

One of the design principles of this work is to promptly react to a lack of transparent knowledge caused by ADC whose focus is on predictive quality aspects. To minimize this gap, we introduced a new approach in this chapter. It identifies missing transparency and complements the existing knowledge base by extracting patterns from arriving flow data that are not already covered by PDB. Furthermore, it constantly attempts to improve the quality of existing patterns. This method is called InDBR. It constitutes an

incremental rule learning algorithm based on our in-DB VPRS model and is intended to implement the integral parts of PBC. Due to the general-purpose characteristics of this pattern builder, its distinguishing properties in contrast to other related concepts can be summarized as follows: as opposed to existing in-DB RST implementations, InDBR is the first approach that addresses the continuous extraction of certain and uncertain decision rules in nonstationary environments. Among other related incremental rule inducers designed for changing conditions, it holds a new bottom-up learning strategy with the potentials to achieve a high inductive leap. In order to obtain this behavior, InDBR assumes that a successful generalization can be leveraged to bring other more specific patterns to the same level as far as sufficient evidence is provided. This can be accomplished with minor effort due to its set-oriented perspective. Moreover, InDBR integrates a distinct partial memory that strives to anticipate concept drifts and class imbalance problems implicitly. Particularly, the latter point is rarely addressed in related literature considering nonstationarity and stream mining.

Apart from these concrete distinctions, in-DB processing has several key benefits. In essence, it brings analytics close to the data and can reduce transport overhead to a large extent. This advantage is exploited in our HFIDS architecture because TDB and PDB are highly penetrated by PBC and so communication costs and data movements between these components are minimized as they can be considered a logical unit. Additionally, modern DB systems are scalable platforms providing efficient data structures and parallel algorithms to process residing data. These assets given by in-DB analytics are in contrast to most related rule learning methods known from the stream mining community. Mainly, they follow a typical processing pipeline that assumes all data to be passed to the rule learner, which manages to update the existing model incrementally in a centralized fashion. Without a doubt, these implementations are lightweight and require minor effort for the processing. However, approaching this way is crucial for real-world scenarios with scalability demands. This can be justified along the following two points: (i) due to their centralized and stiff architecture, a direct application to big data frameworks such as Spark or Flink (see Section 2.3), that spread data to a vast amount of computing nodes, is restricted as processing is bounded to a single machine. (ii) Additionally, available realizations turn out to be single-threaded only, which exposes that an unexpected throughput peak produced by realistic data volumes arriving to the system can justly overwhelm available resources. As a result, unfavorable delays of the processing chain or information loss can arise even though the mining algorithm comprises a very efficient implementation.

These drawbacks outlined in the previous paragraph could be verified during empirical assessments in this chapter where InDBR was tested against the two state-of-the-art top-down rule learners VFDR and G-eRules under drifting conditions. In this comparison, InDBR demonstrated scaling potentials by appending more resources. Yet, its time consumption was still behind by at least one of its contenders. We attribute this poor result to the selected sequential experimental setup dictated by VFDR and G-eRules in which InDBR's true capabilities could not be revealed. Its preferred operating mode is batch-by-batch processing rather than example-by-example treatment as a consequence of its in-DB design. Different insights were obtained from a predictive perspective for all three rule learners supported by the Friedman and Wilcoxon tests. While the any-time

classification performances of VFDR and InDBR were without major distinction, the evaluation further outlined a perceivably high *ABR* of VFDR when ignoring its default rule. In general, this finding has a huge practical impact with respect to transparent and interpretable decision-making because a significant proportion of its classifications are, in fact, black box predictions. Comparing InDBR with its direct competitor G-eRules documents a solid win for InDBR in terms of tentative predictive skills and both carried fairly equivalent refraining characteristics. To complement the analysis of all three learners in imperfect environments, we introduced class noise to our experimental setup in addition yielding similar classification and abstaining tendencies among the learners in contrast to the conventional setting. Nonetheless, an interesting result could be showcased for InDBR virtually maintaining the same tentative classification capabilities in more than half of the benchmark tests. This indicates a stable performance using expressive decision rules without a significant loss of predictive quality despite the presence of noise. The downside of this finding was an increase of the *ABR*, which can be considered acceptable particularly for critical applications that require quality predictions and traceability for domain experts rather than unexplainable prospects. Basically, the same abilities can be awarded to G-eRules. Yet, results were not as explicit as those of InDBR. Contemplating VFDR in this respect is not appropriate because of its extreme *ABR*. Cumulated, more than every second prediction of this learner produced undesired black box classifications throughout our experiments.

Beside these predictive results, a supplemental examination was provided regarding discovery-oriented aspects, which can be counted among one of the first attempts giving insights in this direction especially when considering nonstationary circumstances. This pattern quality assessment disclosed that the model complexity of VFDR is remarkably low followed by InDBR and G-eRules. In terms of the rule length, VFDR induced the shortest patterns in our setup whereas InDBR and G-eRules showed a tendency to produce larger ones that are closer to the data. These sound results of VFDR can be explained along two lines. On the one hand, its high refraining behavior manifests a lazy rule induction process, which in turn tends to keep rules and the rule set itself compact. On the other hand, bottom-up rule inducers typically require more rules to cover the same amount of data and produce larger patterns as opposed to top-down approaches. However, this explanation conflicts with the results of G-eRules, which are close or behind InDBR. Thus, we argue that the generalization and pruning activities steadily performed by InDBR certainly pay off in contrast to the approach taken by G-eRules. In terms of multi-matching, the rates of VFDR and InDBR were alike rank-wise but results of InDBR were more homogeneous throughout the experiments. If we take into account the small rule set sizes of VFDR in addition, that should cause fewer pattern overlappings intuitively, a better performance of InDBR can be substantiated. In this context, the outcome of G-eRules is the poorest where literally every second prediction comprised a potential rule conflict. The final two discovery-oriented measures utilized in our setup were concerned with the rule set coverage and the rule purity. In the former category, InDBR and G-eRules stood out for their solid adaption to most recent data whereas VFDR was far off. This weak outcome of VFDR can be related again to its high *ABR* uncovering that no appropriate patterns were in place to provide a sufficient coverage. Moreover, we should note that a considerable proportion of valid patterns did not cover examples at all. Regarding the purity, InDBR outran both competing

algorithms by a big margin constituting the highest individual rule quality in eight out of nine benchmark tests.

As our main attention relied on transparent decision-making, we devoted our evaluation to pure rule learning algorithms exclusively even though several other non-rule classification approaches are frequently utilized to address concept drifts. For this reason, we want to give a brief report about our experience with two of these algorithms in relation to InDBR. As supplied in [429], InDBR was compared to an adaptive NB model and the state-of-the-art tree learner VFDT with respect to the any-time classification performance in nonstationary environments. In this setting, InDBR won most comparisons or it was close to numbers produced by NB and VFDT. Thus, we rate its predictive performance to be very competitive even when confronted with non-rule learning approaches, which stresses its merit in addition. Moreover, some impressions about the performance of AVFDR can be provided at this stage as an enhanced version of VFDR with explicit drift detection capabilities (see Section 9.2). Even though its implementation was unavailable during our tests, we studied the experimental results in [418, 462], which revealed that VFDR and AVFDR behave fairly similar in terms of misclassification rates in a prequential setup. Beyond that, the evaluation demonstrated AVFDR's rule set is generally smaller on various benchmark data sets. Combining these findings with our experimental experience acknowledges an even higher abstaining behavior when disabling its default rule and, thus, a coarser coverage on most recent data. The reason for this expected result can be partly explained by contrasting the learning strategies of VFDR and AVFDR which are, in fact, identical over large code segments.

Beside the positive results of InDBR, we also want to recap follow-up activities and some of its shortcomings found throughout this chapter. Based on the development progress of InDBR, its current version is only capable to run on categorical data, which implies that a complete discretization of relevant flow and host features has to follow. This is not the only consideration that needs to be taken into account to realize the HFIDS from a practical standpoint. Moreover, InDBR's general-purpose implementation requires additional modifications particularly at the staging table. Processed FVs at PBC cannot be simply deleted as there is a demand for temporal preservation at TDB to serve ADC periodically in terms of retraining. A further issue identified during the realization is concerned with the syntactical and conceptual deviations of modern SQL-based DB systems. In general, these factors complicate a transport of in-DB analytics to other systems especially from a development point of view. Hence, it is not surprising that adjustments or a reimplementation might be indispensable bringing our Oracle implementation to other DB systems. The final drawback is related to the assertion of a unified parameter settings for InDBR. Throughout our experiments, we were unable to figure out such a configuration. We attribute this point to the involvement of different synthetic and real-world benchmark data sets reflecting distinct characteristics. This includes not only the type of concept drifts comprised but their intensity as well. For instance, gradual drifts can occur on a longer or shorter scale for different data populations. Based on that, the ideal capacity of our partial memory varied widely across different data sets in this chapter. This can become even more crucial when the intensity fluctuates over time on a single data stream. For that reason, approaches such as given in [419, 435] involve sliding windows that dynamically contract or expand

according to the current needs, which is expedient and clearly worth to consider. Taken the partial memory capacity as a representative for the parameter fluctuation in our experimental setup, we are convinced to obtain a rather homogeneous setting for the final HFIDS architecture. The argument behind this estimate relies on the aspect that our focus is on a single target domain described by FVs exclusively.

Given all of these aspects, the next chapter concentrates on the concrete realization of the HFIDS, and thereby integrating and complementing InDBR as the essential part of PBC to unveil transparent patterns from arriving FVs. An additional objective is the practical evaluation of the final IDS where InDBR's parameters along with their dependencies are studied in more depth. An important point for this task is the fixation of its operating mode. While, sequential processing was favored herein out of compatibility reasons with other rule learning algorithms, the upcoming chapter presents its abilities in batch mode. Choosing either one is relevant to determine the maximum rule age per class and, thus, impacting its pruning mechanism considerably. In this context, another essential factor of batch processing is to demonstrate InDBR's true scaling potentials, which are addressed eventually as part of the overall HFIDS architecture.

Chapter 10

System Deployment and Evaluation

This chapter assembles all concepts elaborated throughout this work in order to bring up the final HFIDS deployment. We detail the main processing chain to observe arriving flows using a two-step inspection, which is a result of our hybrid detection approach. Moreover, we discuss the incorporation of the pattern building process and rectify several issues that arise when mounting all fundamental blocks in practice. In this regard, the technical implementation has a focus on parallel processing. Consequently, an evaluation must follow where the scalability potentials of the system are assessed. Besides processing capabilities, further aspects are concerned with the behavior under concept drift and which attacks can be identified eventually using flow data as primary entity for the detection. Experiments reveal several important findings. Even though the system showcases moderate detection performances using pure flow data, attack detections can be increased enormously when incorporating traffic statistics. The downside are additional costs caused by preprocessing efforts. Moreover, we compare the HFIDS with other state-of-the-art learning algorithms. In the basic setting, our system shows better performances but when including traffic statistics as well, results become closer. We relate this outcome to the learning setup favoring the other inducers. When porting all approaches to a more realistic setting where labeled data are rare, the HFIDS wins the competition by a big margin. Apart from this outcome, a remarkable qualitative asset of our system is transparency. The HFIDS provides explanations for taken decisions in nine out of ten cases under stable conditions. This is beyond the capabilities of competitive learners.

10.1 Introduction

The foregoing chapters detailed several concrete and important facets on our way to build up the final HFIDS. The problem, however, is that most of these points were investigated in isolation requiring additional refinements in order to resolve potential conflicts. This chapter is devoted to this subject by bringing together all isolated pieces towards a unified technical deployment. Hence, several reconsiderations have to be taken into account. For instance, execution aspects of ADC (see Chapter 8) are not ideal with respect to our parallel processing ambitions and so this intrinsic component of the system architecture has to be revised. In fact, it is divided into two separated modules eventually. PBC (see Chapter 9) has to be integrated into the system landscape as well. This poses a challenge because it rests upon the in-DB paradigm, which conflicts with processing paradigms of other building blocks. In a similar context, a further concern is the pattern matching as it can cause practical issues when using a conventional client-server model to access data from remote DB systems. Thus, problems in that direction have to be sorted out by elaborating alternatives. Apart from the system deployment, we also perform an evaluation in a defined private virtual test environment, which is very competitive with respect to other visualization services that are available commercially. In that testbed, we determine operating points systematically based on different drift scenarios. This is an important undertaking given the amount of variables to be set to control the learning behavior of the system. On these grounds, further tests are realized including an analysis on a real-world data set collected at enterprise facilities in order to examine whether selected operating points hold beyond the drift scenarios. Additionally, the system is assessed in terms of detection capabilities and to which extent it can scale when we increase throughputs and hardware resources.

According to the mentioned points, the remainder of this chapter is split into three sections. First, we assemble all components with technical details and rectify issues towards the deployment of the HFIDS (Section 10.2). Second, the mentioned evaluation is conducted to outline the practical assets of our proposed system (Section 10.3). Lastly, a summary is provided concluding this chapter (Section 10.4).

10.2 System Deployment

This section examines the system deployment with an emphasis on technical details. Initially, we sketch the processing chain that incoming data have to pass (Section 10.2.1) and discuss how the pattern building functionality is incorporated (Section 10.2.2). With these two fundamental blocks in place, all concrete stages comprised by that framework are described: we present required preprocessing steps (Section 10.2.3), the implementation of the pattern matching engine (Section 10.2.4), the modules of ADC (Section 10.2.5) and the alert generation and data persistence (Section 10.2.6). As such, a global picture is outlined to give a final wrap-up of the concluding HFIDS (Section 10.2.7).

10.2.1 Main Inspection Pipeline

According to the conceptual architecture proposed in Chapter 5, each and every IPFIX biflow arriving from the network system to monitor has to traverse several stages before they are persisted at TDB. We name this set of stages "main inspection pipeline" (MIP) because it mainly consists of PMC and ADC, the centerpieces of the two-step hybrid detection approach. In the first step, PMC tries to match incoming data with existing patterns. If this is not possible, for what ever reason, ADC comes into operation constituting the second line of inspection. Both components expect FVs as input with an option to be enriched using host events and/or traffic statistics controlled by choosing one of the four FFCs (see Section 7.3.2). As such, several preprocessing activities are required turning biflows and supplemental data into FVs. This task is addressed by DPC. The last stage of MIP, which comes right after ADC, is concerned with the persistence of processed FVs with associated inspection results and the generation of alarms, if any. This duty is carried out by "AM and persistence" (AMP).

Given this pipeline, it becomes quite intuitive that processing has to take place sequentially. If we think of a single-threaded solution to implement that chain, one flow is passed through at a time causing all other flows to wait. This is not an issue as long as processing completes before the next flow arrives. In practice, however, many flows arrive concurrently at high velocity dependent on the utilization of the network system to monitor and so the sketched solution is easily overwhelmed causing enormous delay and data loss ultimately as buffer mechanisms exceed sooner or later (also see discussion in Section 9.5). Hence, there is a demand for parallel processing where multiple MIP instances can coexist to increase throughput rates. This is when stream processing engines come into play (see Section 2.3.3) because they provide native solutions for such circumstances. Several of those software frameworks come to mind but choosing the right one is not a trivial task as all enclose their specific advantages and disadvantages. To figure out which is suitable for our use case, we benchmarked Storm and Flink in 2016. Both are renowned for low latency stream processing, which could be confirmed by our analysis. Yet, our experiments also revealed that Storm often crashed at runtime due to back pressure and so it became very difficult to determine whether those crashes were a result of its limited stability at that time[1] or caused by too much data load. Flink in turn demonstrated a more stable behavior. Additionally, Flink natively follows the "exactly-once" principle and has a more active development community than Storm that already passed its zenith in our opinion. All of these aspects finally guided us to select Flink over Storm to implement the MIP. To compensate potential data bursts, we also incorporate the message-queuing system Kafka that serves Flink as input data pool.

The sequence of all four stages involved in MIP is outlined in Figure 10.1. Note, details of each individual stage are omitted by purpose as we intended to present a high-level view in this Section 10.2.1. In upcoming sections, we detail all relevant internals consequently. This also includes the coupling between both MIP and the pattern building capability justifying a modified version of the kappa architecture (see Section 2.3.2).

[1] Note that stream processing software solutions are under constant improvement and so it is very likely that this issue is already resolved in newer releases of Storm.

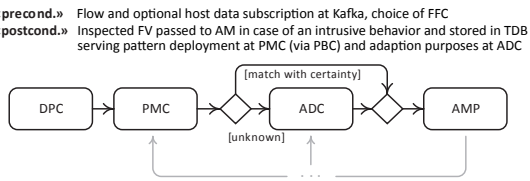

FIGURE 10.1: High-level view of the MIP as activity diagram to prepare and assess the arriving stream of flows with optional host data using Flink

10.2.2 Incorporating the Pattern Building Process

In the previous Section 10.2.1, we gave a brief overview of the main stages required to inspect arriving FVs in an online fashion. However, we neglected details of how to make sense of the labeled output stream in such a way that the HFIDS can benefit from performed predictions. A key aspect in this respect is the construction of patterns that are intended to be looped back to MIP in order to improve subsequent classifications in a transparent manner. This section outlines parts of this matter. Concretely, we describe the underlying technical platform to implement PBC and discuss specifics that have to be taken into account in addition.

Given the basic design of the HFIDS proposed in Section 5.5, two data repositories are required as intermediate storage, i.e. TDB and PDB. Recalling that TDB's main task is to accept consumed FVs from Flink and PDB accommodates deployed patterns, a practical problem arises based on this layout: both DBs are frequently penetrated by PBC to build up quality patterns, which is why PBC's footing relies on InDBR (see Chapter 9). This permits to organize all three components in one single DB system using the in-DB paradigm. Certainly, this can pay off reducing communication overhead but also requires a scale-out DB architecture to be on par with the distributed processing capabilities of Flink. Thus, two basic concepts are worth to consider for this duty, i.e. either a shared-nothing or a shared-disk architecture (see Section 2.3.4). In this regard, the general amenity of a shared-nothing cluster is the high autonomy among individual DB nodes enabling a high degree of distributed query processing. A prerequisite for this functionality, though, is to find a suitable data partitioning strategy and even if a strategy is straightened out, access patterns may change rendering the current configuration ineffective. In our context, practical impressions applying shared-nothing solutions such as Greenplum and Postgres-XL are two-fold. On the one hand, the insertion of FV batches to TDB demonstrates no major concern either combining range, hash or round-robin partitioning. On the other hand, high query latency are noticeable in our setup mainly caused by InDBR. We relegate this circumstance neither directly to the potentials of available shared-nothing solutions nor to InDBR but to the absence of a proper horizontal partitioning strategy for our data preventing a timely deployment of patterns eventually. Based on this setback, we also evaluated the shared-disk system RAC, which gets over the partitioning issue. Both, FV imports from Flink as well as parallel query processing showed promising results during extensive testing. This is why RAC is our choice to incorporate the functionality of PBC, PDB and TDB ultimately. To get a general overview of a typical RAC installation, Figure 10.2 sketches the main

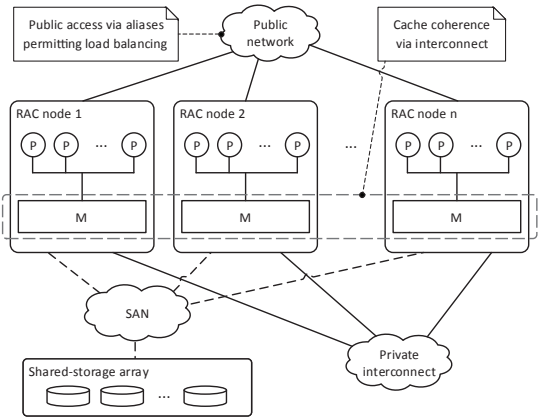

FIGURE 10.2: Schematic view of a typical RAC comprised by DB nodes with multiple processors (P) and main memory (M) sharing the underlying DB content via a SAN which is utilized to implement PBC, PDB and TDB as processing and storage backbone

components where Flink basically communicates to RAC via the public network and internal distributed data processing is carried out via the private interconnect. To ensure RAC can deal with the FV ingestion from Flink, we tuned the DB system according to official guidlines postulated in [463].

Having selected RAC as technical platform, further particularities need to be discussed. This include modifications to the general-purpose algorithm InDBR serving PBC and its distinction to classic misuse detection. In this respect, we already stated practical performance concerns when instructing InDBR to process input data on an example-by-example basis, which would be a FV-by-FV treatment in our problem domain to be more precise. Hence, we apply its preferred operating mode throughout the remainder of this chapter, i.e. incremental learning based on consecutive FV batches (see Section 9.4.4 and 9.5). This behavior is illustrated in Figure 10.3 (a) where FVs at TDB (arriving from Flink) are constantly consumed in batches to produce or update patterns. After a batch is worked through, corresponding FVs in TDB are marked as processed and PBC rechecks for new batches consequently. If no new data are present, a wait mechanism is introduced that resumes work after a short period of time. Having said that, it should be clarified that TDB and PDB are not independent DBs. Technically, they rather refer to conventional tables hosted by RAC. Specifically, TDB and PDB portray the staging table T and the rule set R introduced as part of InDBR's table schema in Figure 9.3. Anyhow, we stick to this naming convention for the most part and continue to reference those tables with their symbolic aliases for the sake of simplicity, i.e. TDB and PDB instead of T and R. In addition, it should be stated that the treatment of processed FVs in TDB deviates from the standard behavior of the general-purpose implementation. Normally, InDBR would delete consumed entities from the staging table right away but this functionality is delegated to ADC instead. This adjustment has practical reasons and is further address in Section 10.2.5. Hence, another control mechanism has to be introduced signifying processed FVs from a pattern building perspective. Therefore, we

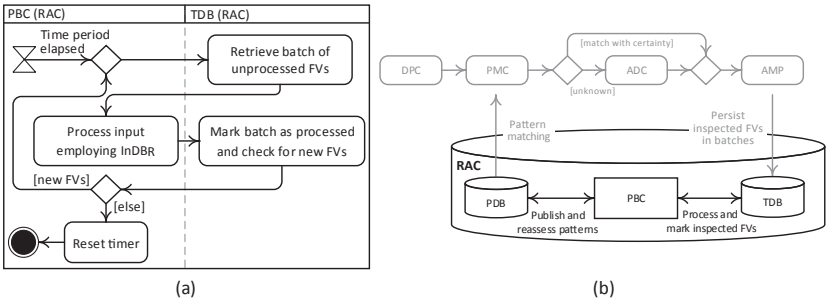

FIGURE 10.3: The pattern building process deployed in a scalable RAC infrastructure: (a) FV ingestion with periodical processing and (b) a broader view of the DB back-end to reintegrate patterns into MIP using in-DB analytics

revise the *status* in TDB's attribute schema by establishing the new integer value 2. It indicates that corresponding FVs holding this flag are already processed by PBC and should not be reconsidered. Consequently, this status is assigned for all FVs that passed the pattern building process. Moreover, an update mechanism has to be considered for those cases where ADC identified an anomaly or AM resolved a new evolved cluster that is associated to anomalous FVs, which is not covered by InDBR out-of-the-shelf. Realizing this functionality requires another status, i.e. the integer value -1. Irrespective of their earlier status, it signifies the necessary reassessment of FVs. This modification inside the algorithm is straightforward though. During the consumption of a new input batch, PBC not only involves FVs with status 0 but also those with status -1. Hence, it seeks for entities in its partial memory W that correspond to FVs with status -1 in the batch and updates their class labels accordingly, while the input with status 0 is just appended to W. All other steps basically remain unchanged such that the complete batch is finally propagated to PDB. Yet, a special situation evolves if no associated entity can be found in W anymore for incoming FVs with status -1. This requires PBC to perform a supplemental step. It is comprised by determining patterns within PDB covering those FVs to rectifying their consequent eventually, if required. This update mechanism implicitly leads to another characteristic of PBC, i.e. its ability to create patterns for any class whether being "malicious", "benign" or "anomalous". This significantly extends classic misuse detection, which is also proposed in [257]. In our context, several benefits can be deduced. First, this feature comes free of charge due to the general-purpose implementation of InDBR and so no major code adjustments need to be considered. More important though, a relaxation of ADC can be accomplished, which reduces prediction costs at that side, black box classifications and thereby enhancing transparency aspects at the same time beyond the monitoring of attack data. In this respect, an interesting point is the handling of concept drift for legitimate flow traffic. By definition, such scenarios refuse the creation of benign patterns because normal footprints in a network environment change sooner or later rendering these patterns ineffective (see Section 5.2). However, this issues is less critical when patterns become temporal. This transience is supported by InDBR using its underlying aging facilities. We challenge the effectiveness of this feature under different drift scenarios throughput this chapter and figure out operating points for the pattern aging per class.

Having discussed all those specifics to couple the pattern building process with MIP by heavily employing in-DB principles using RAC, a broader picture of this endeavor is outlined in Figure 10.3 (b). To this point, technical details about the pattern matching itself have not been debated. We defer this discussion to the separated Section 10.2.4 instead.

10.2.3 Data Preprocessing Steps and Correlations

We conducted an analysis based on flow features and identified several essential attribute sets considering flow data exclusively or their combination with traffic statistics in Chapter 7. In addition to these distilled reducts, log events residing on host level were also incorporated as supplemental data source. Using flow data as baseline, four different FFCs emerge with the potentials to serve a proper intrusion detection process, i.e. F00, FH0, F0W, FHW. Our HFIDS implementation supports all of these FFCs in the sense that one of these FFCs must be chosen as mode from the very beginning and so the IDS produces exactly one FV for one incoming IPFIX record. Yet, those FVs can not embody the attribute structure of a single reduct alone further recalling that one key of ADC's ensemble diversity relies on selecting different reducts for different members (see Algorithm 8.3). As a matter of fact, each FV must convey the union of all exposed reducts, i.e. $R_1 \cup R_2 \cup R_3$ dependent on the chosen FFC mode (see Table 7.4). In this technical light, a FV carried through the system actually refers to a "super FV" from which ensemble members can pick and choose their individual FV according to their assigned reduct specification. Unlike the application of this concept for ADC's supervised component, its unsupervised component has a rather global view and choosing one reduct over another might be seen inappropriate even though one of them is already sufficient for clustering purposes. Not to be spoilt for choice, we define the input of our chosen clustering method DBStream to rely on super FVs per FFC. Similar motives can be acknowledged for the creation of patterns and so PBC shall also rest upon the complete input source rather than on a single reduct. Combining this line of thought leads to several preprocessing activities ingress data have to undergo in order to build up apposite FVs to be classified by the system eventually. Encapsulated by the DPC activity (see Figure 10.1), they can be narrowed down to the following three steps implemented using Flink:

(i) Creation of basic FVs with IPFIX data and feature extraction

(ii) Correlation of FVs with available host information, if any

(iii) Transformation of features (i.e. normalization and discretization)

In step (i), Flink listens to a dedicated Kafka "topic" serving newly arriving IPFIX records from the network infrastructure of interest. For each of those records, an FV is created with native flow data while irrelevant information is discarded. Moreover, this steps involves the extraction of several non-native flow features associated to a FV, i.e. the flow state, SC and traffic statistics (see Section 7.3.2). While all of these consecutive actions are implemented by separate Flink operators, all of them can be merged into one "operator chain" (see [464]) except for the traffic statistics which we discuss shortly. This chaining is possible because neither the creation of a FV nor the extraction of

the two mentioned features (see flow state and SC) is related to other entities. The feature extraction simply relies on lookup tables and information already residing in the flow from which to create the FV. This independence among those subactivities provides performance benefits as all chained operations can be co-located within the same thread. This avoids context switches and unnecessary data movements ultimately. In case of the traffic statistics, further operators are required though. Concretely, we have to count the number of adjacent flows to the same SC or the number of concurrent flows to the same destination address as base setting. Such calculations in turn are not directly feasible given the distributed nature of the stream, which justifies why involved operators cannot be part of the earlier described operator chain. In order to comply with the semantics of the traffic statistics, the data stream has to be repartitioned along the two features SC and destination address instead. Flink supports this reorganization such that subsequent operator instances consuming the stream can expect FVs to arrive from the same SC or the same destination address, i.e. a "keyed stream" (see [465]). Apart of this repartitioning aspects, the counting requires stateful operators which we realize using customized windows. For both keyed streams, a time-based window of ten seconds and a sliding window keeping the last 200 adjacent flows is implemented. However, as opposed to original definitions (see Figure 7.2), some pitfalls may occur in practice. For instance, we observed during testing that under high pressure (typically resulting from legitimate high network utilizations or DoS attacks), the time-based window can grow rapidly to unmanageable sizes degrading Flink's performance. Another issue arises with the sliding window where, in some cases, FVs can reside in a window for several hours or even days. Obviously, both points are not expedient requiring further constraints. To bound both windows, we only permit each time-based window to grow to maximum size of 1000 FVs whereas the sliding window is restricted to hold FVs from the last 60 seconds exclusively. After the computation of the traffic statistics, step (ii) is initiated. This step simply can be understood as the fusion of the prepared FV stream just assembled with a concurrent data stream sourcing host information from a second Kafka topic. Yet, the fusion of both streams is not straightforward given the fact that elements of both data streams are not produced simultaneously and so the notion of a conventional join operation known from DB theory (see Section 4.3) does not suffice solely. Additionally, our realization takes into account delayed arrivals of host information and even those situations where such information is not available at all. To obtain this functionality, a "keyed global window" (see [465, 466]) is deployed. As the name infers, two components are included, i.e. a keyed data stream and a window for each of those keys. As opposed to the earlier keyed streams, the keys for the global windows are a composition of IP addresses and ports this time, which permit to associate existing FVs and relevant host events. In this respect, each window instance can be considered as temporary buffering mechanism with a specific time-based triggering policy of 30 seconds. Every time a window triggers, host data are appended to a FV as long as inherent timestamps overlap by at least 60%. However, we cannot ensure that host data become available in a timely manner due to potential delays arising in practice. For this reason, we keep FVs in each window for two triggering cycles right before they are released either with a successful correlation or without. Combined, this means that FVs are buffered in global windows for a maximum time period of 60 seconds, which basically refers to an "outer join" behavior in DB terminology enhanced with a delay mechanism. Note that host data which can not be associated with any existing FV are purged after four triggering cycles.

The sequel of the correlation is a round-robin rebalancing of the data stream followed by some final actions involved in the main DPC activity. These actions are comprised by step (iii) and are mainly concerned with feature transformations implemented by a chain of Flink operators. The reason for those conversions is two-fold. So far, the existing feature inventory encapsulated by each FV consists of a mixture of numeric and nominal data types. While many learning algorithms can deal with these raw attributes in our setup, others cannot. This is particularly valid for those algorithms using the Euclidean distance or similar measures relying on some normalized input space[2] like DBStream. In that sense, two duties are notable to obtain such a normalization, i.e. converting categorical attributes such as SC or flow state to a numeric data type and bringing all numeric features to a unified scale. These points are carried out using standard methods that are commonplace in network security research. In order to turn categorical feature values to numerical ones, we employ a form of "metric embedding" often called "one-hot encoding" (e.g. [23, 327]). This technique basically produces m synthetic features a_j with domain $V_{a_j} = \{0, 1\}, 1 \leq j \leq m \in \mathbb{N}$ for a given categorical attribute a where $|V_a| = m$, i.e. a m-valued nominal attribute. As such, a_j holds the value 1 if a's actual value is the j-th element of V_a or it is set to 0 otherwise. To scale all numeric attributes, a slight modification of the conventional "min-max transformation" (e.g. [467]) is utilized bringing their numeric values to the interval $[0, 1]$, which justifies the normalization procedure. The second aspect for a data transformation rests upon PBC and PMC requiring a discretization of numerical attributes (see Section 9.3). As a solid discretization was already conducted in Section 7.3, we reuse those results to create lookup tables for each feature, which are applied in the corresponding operator chain. This way, most numeric values can be safely mapped to a discrete value ranges, while we assign the value "unknown" in those cases where no appropriate lookup table entry is found during runtime. We would like to state at this point that the feature transformations are not meant to replace their original feature counterpart. Instead, all features whether original, normalized or discretized remain in place and are hosted by the respective FV. To summarize our discussion of involved subactivities comprised by DPC, a generic breakdown of it is provided in Figure 10.4. Parallel to this breakdown, the figure also includes a schematic overview of required stream reorganizations between operator chains (see gray circles) per FFC. In this regard, the most elementary data flow is produced for F00. In contrast to FH0, F0W and FHW where the data stream has to be heavily repartitioned due to window computations (see crossing stream paths), F00 is not invoking such activities at all. As a result, the partitioning as well as the order of FVs can be preserved from a processing perspective (see straight stream paths). Disregarding the different complexities of those data flows, it should be emphasized that all chosen Flink operators in all preprocessing pipeline variants are highly parallel and distributable via multiple computing nodes delivering scale-out capabilities. Yet, it is worth knowing to which degree each FFC contributes in terms of speedup and data throughputs. Therefore, we give practical insides about scalability aspects in Section 10.3.

[2] Several distance measures are highly susceptible to different feature scales and would prioritize high-valued features over smaller ones. This is generally not a desirable objective and requires some sort of normalization (e.g. [321]).

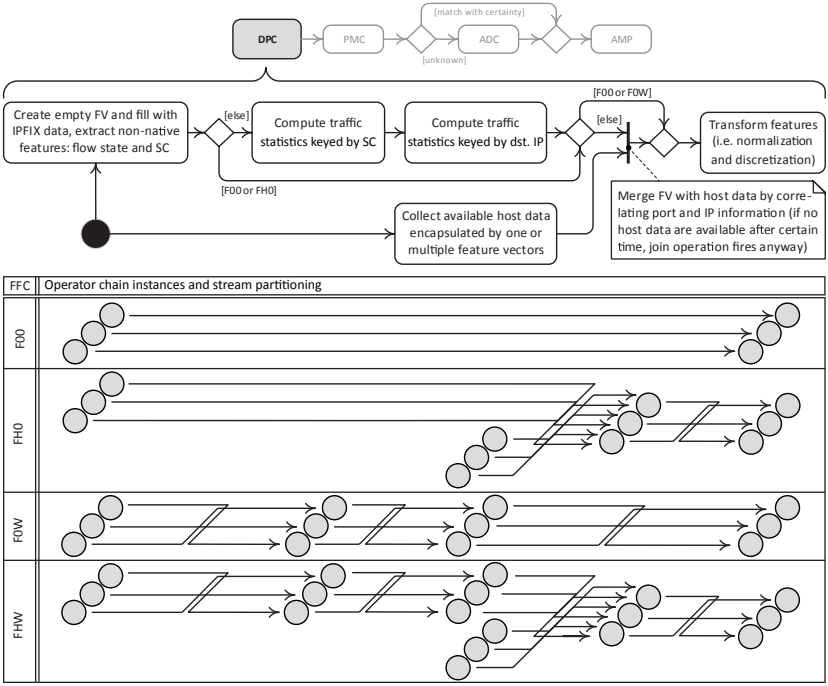

FIGURE 10.4: Breakdown of the required preprocessing steps at DPC dependent on the FFC mode shown as activity diagram (top) and schematic overview of DPC's distributed stream partitioning among parallel Flink operator chain instances exemplified by a parallelism of three (bottom)

10.2.4 Pattern Matching and Distribution

In Section 10.2.2, we described the in-DB process to deploy patterns at PDB based on previously classified FVs residing in TDB. This is controlled and conducted by PBC, which, on the other side, follows the functioning of InDBR with minor modifications. In order to leverage this pooled knowledge inside PDB in the sense of a transparent classification process for unseen FVs, yet a mechanism must be devised and implemented to match patterns efficiently from within MIP. This task is conducted by PMC and detailed in this section. Starting with a general overview of the pattern matching engine loosely coupling PMC and PDB, practical concerns are outlined inherent with this approach. Against this backdrop, several technical remedies are discussed, which lead to an optimized solution permitting to match patterns in a distributed manner.

As PDB is hosted by RAC, a straight solution to make patterns available to PMC is to employ the SQL application programming interface as convenient communication channel between the two components. In this setting, Flink nodes act as clients sending queries to RAC that in turn processes requests and determines whether patterns exists

for a given FV. Further developing this query-based pattern matching approach, three sequential key activities can be identified. The first activity addresses the query building phase initiated within Flink. It comprises the preparation of a SQL statement based on condition feature values embodied by the current FV in question and submits the query. RAC takes care of the efficient query execution balancing the load among its nodes through the next activity. Such a query prototype is outlined in Listing 10.1 with condition features a_i and corresponding values $v_i, 1 \leq i \leq m \in \mathbb{N}$ seeking for both most specific and generalized active patterns.

SELECT rId, a_1*, ...,* a_m*, d, createdAt, lastModifiedAt, delta, alpha*
FROM R AS PDB
WHERE $(a_1 = v_1$ *OR* $a_1 =$ ‘$*$’$)$ *AND ... AND* $(a_m = v_m$ *OR* $a_m =$ ‘$*$’$)$ *AND*
$isActive = 1$

Listing 10.1: Generic SQL query to match patterns in PDB

This generic search is implemented reusing the $*$-symbol introduced in Section 9.3.4. Recall that all patterns are stored under a uniform and fixed table schema, this symbol models the absence of literals in case a pattern is more general than the most specific. Having outlined the query-based pattern matching from the preparation to the execution, results returned to PMC need to be managed consequentially. At this stage, it must be quoted that the pattern search may involve a result set of active decision rules that cover the corresponding FV's characteristics. Thus, three notable cases must be distinguished, i.e. the set may be empty or may contain one or multiple rules. While extensive experiments demonstrated that the first two situations cover most circumstances predominantly, the latter can not be ruled out in the general case (see multi-matching). This state of affairs is crucial because it may cause ambiguity from both a classification and interpretability perspective requiring adequate amendment. Therefore, the last activity comprised within PMC examines the query outcome to finally tag a FV “malicious”, “benign” or “anomalous” based on majority voting[3]. In the event of a decision tie or no rule fires at all, an intermediate classification label is utilized for a FV, i.e. the “unknown” flag, signifying the demand for further assessment at ADC (see Section 10.2.5). Yet, if at least one rule exists that refers to the malicious class, obviously the FV in question is suspicious and needs revision at AM. Hence, all rules involved in the decision-making process are associated to the FV serving follow-up actions. A complete picture of PMC with all intended activities is illustrated in Figure 10.5.

Even though these building blocks represent nothing but common practice to interact remotely with DB systems, this client-server model may cause severe performance problems. Particularly, this is the case in environments where the DB system is accessed continuously, which applies to our stream-based processing approach. To exemplify such critical situations, let us assume that the Flink pipeline runs on a single dedicated machine while ignoring PMC for a moment. In this simplified setup, Flink certainly is able to process several hundred up to few thousand FVs per second. Transferring these throughput rates to the PMC setup entails that at least the same amount of queries are

[3] More advanced techniques can also be applied such as weighted majority votes based on rule confidences. However, since we expect the cardinality of the result set to be between 0 and 1 for most cases, further elaboration is omitted at this stage.

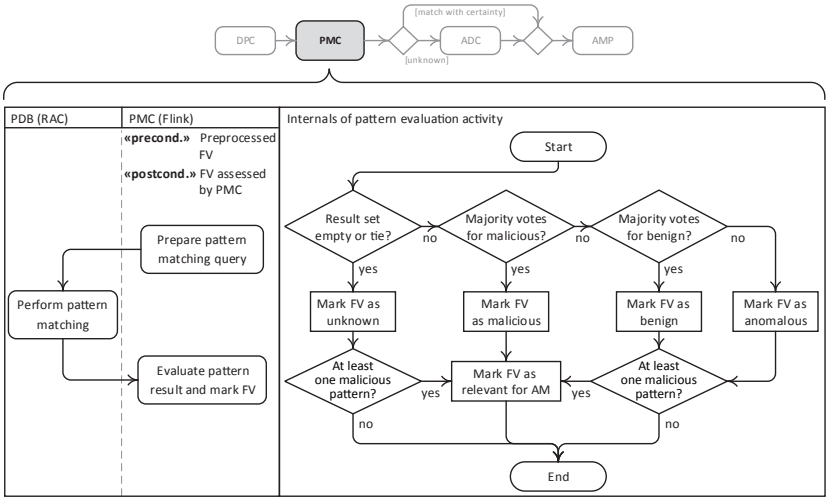

FIGURE 10.5: PMC processing: its building blocks as activity diagram (left) and the evaluation of the pattern matching result per flow as flowchart (right)

sent to PDB in that time frame each requiring low latency at the DB end in order to keep up the speed of the application-tier. This fact in turn sets high expectations for modern disk-based DB systems such as the utilized RAC, which can be barely met as queries commonly touch storage to retrieve data irrespective of distributed and highly parallel execution plans chosen by the query optimizer. Thus, increased latency can be implied compared to the stream-based processing of Flink mainly operating in memory. This speed discrepancy between PMC and PDB is further fueled considering that the Flink program is executed not only on a single machine but within a cluster targeting scale-out demands ultimately exceeding the capabilities of disk-based DB systems for the pattern matching task. These mentioned performance gaps between DB systems and applications often can be compensated employing batch processing in order to work through multiple records in one pass (see Section 10.2.6). Unfortunately in the special case of PMC, FV-by-FV processing is an obligatory step, which needs to be addressed to eradicate the speed mismatch. Thus, a reasonable line of thought is the orchestration of a DB system's buffer management (e.g. [139, 190]) caching frequently penetrated data blocks in memory. This way, access time to the data can be reduce but even sizing buffer caches appropriately to fit all the data still cannot overcome the DB's basic assumption, i.e. disk-based data residency. Its engine is optimized to serve this primary purpose requiring more machine instructions to manage buffers and data locations among others, which produce overhead in general (e.g. [468, 469]). Additionally, few options are supplied to control cache hits and misses because most of the buffer functionality is governed by DB internals and heuristics further complicating the utilization of the buffer cache solution to close the speed gap. Notable alternatives are "memory-optimized tables", a recent option anticipated by Oracle and other manufacturers as part of their disk-based DB systems (e.g. [470, 471, 472]). Compared to the sole buffer cache solution, they get over the mentioned management costs and provide supplementary data structures and

techniques particularly designed to serve analytical queries. Yet, adopting this approach to our setup improves response times only marginally and obtained latencies are still too high in order to cope with the speed potentials of Flink. In addition to these stated optimization strategies inside the DB system, a further point can be made referring to the network communication between Flink nodes and RAC adding significant costs to the pattern matching task. These should not be underestimated recalling that thousands of queries are sent simultaneously to the DB system and need to be assimilated instantly. To sort out issues in that direction, patterns need to reside close to MIP implying the incorporation of a local cache mechanism. This could be achieved using an adequate memory-resident data structure deployed nearby or inside the Flink program comprising the set of all active decision rules obtained from the PDB. However, such an approach also needs to take care of cache consistency aspects including replication activities to enclose the latest version of patterns. Certainly, this can be considered a non-trivial undertaking neglecting naive refreshing techniques where the entire PDB is copied locally in a consecutive fashion. Hence, an implementation from scratch is discouraging. A more reliable opportunity to pursue this prospect is the utilization of an existing cache engine. Such a framework is supplied as part of Oracle's TimesTen (TT) [473, 474], an in-memory relational DB system, which jointly works with RAC. This framework is called TT Application-Tier DB Cache (TT cache) [468, 475] and is particularly designed to serve performance-critical data access at the application layer. Unlike other cache engine alternatives such as Hazelcast[4], Memcached[5] or Oracle Coherence[6], data remain in their native relational domain with many of its data management capabilities. These include transactional logging and query optimization fully supported by direct in-memory data access patterns. Additionally, it incorporates efficient data replication methods between in-memory and back-end DB. Thus, TT cache is a convenient fit to our caching problem from a functional perspective. It is applied throughout the remainder of this work because it copes with speed demands of the Flink program empirically examined during diverse long-running tests (see Section 10.3). In what follows, technical details are provided highlighting the most important implementation aspects.

The basic idea behind the relational cache engine in the context of the HFIDS architecture is outlined in Figure 10.6 contrasting the conventional remote pattern matching and the distributed solution where each node of the Flink cluster is equipped with a TT installation representing a local cache of PDB. For the sake of brevity, we call it "local PDB". Following this approach, rule matching is performed locally reusing the query of Listing 10.1, while the management to create, update and delete decision rules is still controlled and maintained by PBC and PDB respectively. Technically, this can be accomplished using "cache groups" created inside TT. These define the schema and data to be cached from the remote DB and can be either declared global or local. Global cache groups refer to cached data shared across multiple TT systems forming a "cache grid" whereas a local group caches data alone without any interaction among a TT cluster. Based on that, local cache groups are utilized in our case to avoid extra effort to manage data locations of cached instances, which is a demand for global cache groups. Moreover, the intended cache solution refers to a rather simple approach where the local PDB is

[4] `https://hazelcast.com`
[5] `https://memcached.org`
[6] `https://oracle.com/de/middleware/coherence/`

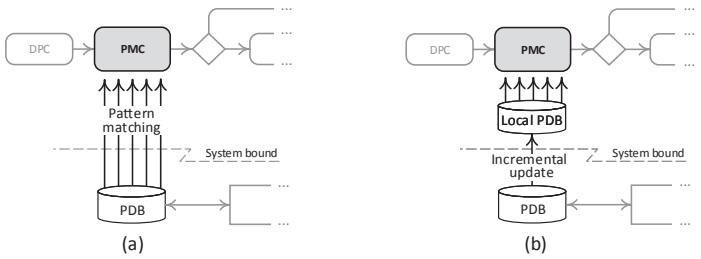

FIGURE 10.6: Conventional and distributed pattern matching approach: (a) the plain setting producing high i/o to match decision rules and (b) the proposed distributed solution with local in-memory PDBs at each Flink node updated incrementally

not updatable at least from an application-tier perspective admitting to configure those local cache groups to be read-only. In this respect, each group essentially reflects an unmodifiable copy of PDB's active decision rules and can be directly accessed using SQL from within the Flink program with all the speed advantages mentioned. This can be accomplished utilizing a special direct driver connecting application and TT without emitting any kind of inter-process communication as data is loaded into the shared memory segment [468]. Given this local setup applied to each individual Flink node, a final concern points to the consistency between the local PDBs and recent changes residing in the remote PDB. This can be sorted out by instructing the cache groups to refresh periodically supporting two modes, i.e. a complete reload or an incremental update of the data. Since it can be assumed that a large body of patterns remain stable during update periods in the long run, it is preferable to refresh the local PDBs in a incremental manner rather than accepting huge data loads causing unnecessary network traffic. This feature is efficiently implemented using a change log table and a trigger inside RAC keeping track of recent changes. This way, an outer join can be utilized to determine the differences between cache group and back-end DB. With these technical remarks, the desired cache functionality is exposed, which is both simple and scalable at the same time ultimately permitting to distribute and match patterns in the context of stream and relational processing. Based on these results, the left part of Figure 10.5 needs revision and can be replaced by Figure 10.7 incorporating the local PDB and the remote PDB with their data replication process.

10.2.5 Anomaly Detection and Warm-up

The basic design of an adaptive anomaly detector has been worked out in Chapter 8. In this section, we tailor and concretize this vision closer to our system architecture with respect to MIP and storage back-end realized by RAC. Additionally, we propose a simple methodology how to initialize ADC with respect to the specifics of the underlying network infrastructure to safeguard. We refer to this phase as "warm-up" right before the system can be turned operational.

The conceptional phase to build up ADC was driven by seeking for a solution addressing nonstationary environments caused by novel attacks, evasions or changes of legitimate

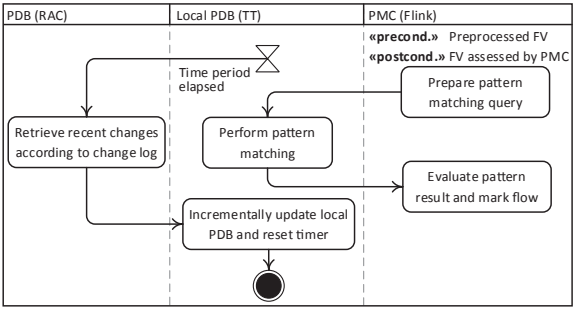

FIGURE 10.7: Revised PMC processing with local PDB and incremental update from the remote PDB illustrated as activity diagram

data all stoked up by class imbalance. Therefore, an internal data buffer $\mathcal{D}$ was tightly embedded in the classification process serving adaption purposes. When it comes to the concrete technical deployment of this solution within MIP where ADC takes over classification in cases PMC is uncertain due missing patterns or ties, the placement of $\mathcal{D}$ is sensible though. As the data buffer is a central constituent, this setup directly conflicts with the distributed nature of Flink where operator subtasks are expected to classify the data stream of FVs in parallel. Additionally, $\mathcal{D}$ would compete with TDB in the sense that both maintain the same data. From this perspective, it is a reasonable line of thought to decouple the functionality of ADC into two separated units, i.e. an "online" and an "offline" component. The main purpose of the online part is to exclusively take care of the classification using the ensemble H inside the Flink program where any FV passing this frontier is marked as "relevant for AM" if H classified it as "malicious". The offline component in turn leverages TDB as data buffer to periodically assess H's performance and recomputes weights or substitutes the ensemble if necessary. Required updates of H are injected to the online part using conventional "object serialization" whereas test sets for H's evaluation are sampled from FVs in TDB originally classified by PMC or ADC's online version. That said, it should be stressed that only FVs are considered where ADC's confidence exceeds ψ_{min}. The clustering is also an essential part of the offline unit taking arriving FVs at TDB as input to maintain its underlying model and if new evolving clusters are identified corresponding FVs in TDB are relabeled as anomalous and marked to be reassessed by PBC (see Section 10.2.2) and H respectively. By analogy to the input data consumption of PBC, the processing of the clustering basically works the same (see Figure 10.3 (a)). Yet, the exception is that FVs comprised within receiving batches are consumed sequentially according to their arrival, i.e. eId in TDB's table schema. A further point of the original ADC is to keep track of data proportions by a cleanup routine. This functionality must be ported with care to our current setup because data in TDB are inherently shared between ADC and PBC. As such, we can only delete FVs from TDB safely if they are already processed by the clustering and PBC, which would produce data loss otherwise. Having discussed both the supervised and unsupervised part of ADC's offline component and the model update for MIP, this restructuring obviously sorts out the mentioned concerns, thereby fitting our HFIDS architecture. In this respect, an additional issue is the incorporation of AM

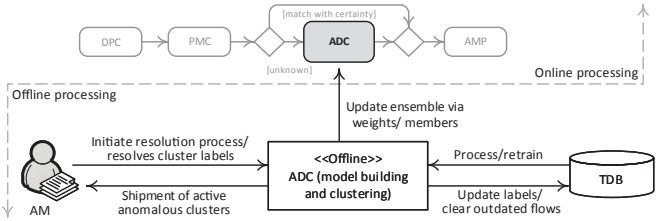

FIGURE 10.8: ADC online and offline processing to permit adaption

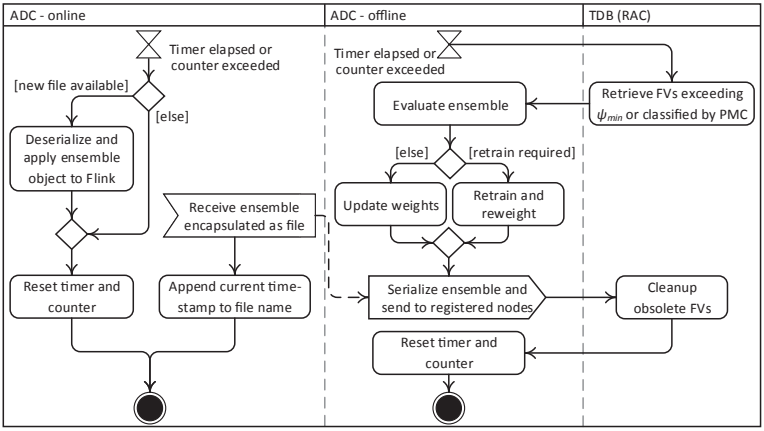

FIGURE 10.9: Recurrent update mechanism of ADC at runtime illustrated as activity diagram

that is intended to interact with the offline part where identified anomalous clusters are revised by administrative staff to determine their real GT. This resolution process is not initiated by the HFIDS but by system operators. Once triggered, ADC extracts a list of all current anomalous clusters and delivers them to AM. After their labeling, these clusters are reverted and ADC seeks for corresponding FVs in TDB to apply the resolved labels. If this action is not possible anymore because FVs were cleared prior to this (see cleanup routine), ADC straightens out such circumstances by taking advantage of the cluster centers converting them into synthetic FVs. These are upsampled according to the actual cluster size and appended to TDB such that both PBC and H can adjust their underlying models respectively. A summary of all those activities is outlined by Figure 10.8 whereas Figure 10.9 provides insights to the update of ADC's online component

With these aspects in mind, a crucial question arises how our HFIDS is actually adjusted to the network system from scratch. Each network site conveys individual particularities an NIDS must learn and be aware of in order to operate effectively in practice, which is especially true when detection partially or completely rests upon supervised ML (see argumentation in Section 6.1 or 8.1). In our case, this familiarization with the network environment corresponds to the initial phase where ADC has to be trained. The inherent

problem with this setup is that characteristic legitimate data can be skimmed right off the network system but GT availability cannot be presumed in general and malicious data may also be comprised to render meaningful models. To take up this challenge, we propose a very simple warm-up mode where our HFIDS is not operational. In this phase, it listens to the network system and creates FVs using DPC, which are intermediately persisted inside RAC and labeled as "benign". From there, these data are salted with data from our broad attack repository NDSec-1 to finally obtain a training set. This training set is then utilized to initialize ADC for both its supervised and unsupervised component right before the HFIDS is ready to monitor the network landscape. However, this methodology assumes that live captures for the training are attack-free, which cannot be guaranteed always. We, therefore, suggest to use a conventional NIDS such as Snort or Suricata[7] in addition to eradicate potential malicious activities from the training set beforehand or to explicitly incoporate them once uncovered.

10.2.6 Alert Management and Persistence

The last step comprised by MIP is concerned with two basic tasks, i.e. the alerting of suspicious FVs and the persistence of classified FVs at TDB. In the Flink program, both duties are implemented by splitting the FV stream. Technically, this can be understood as if the stream is sent to two Flink operations concurrently where the first operation determines if any FVs is marked as "relevant for AM" and in such a case directed to AM. In all other cases, FVs are simply discarded. The second operation, on the other side, also consumes the entire stream but without filter functionality. All FVs are inserted to TDB with corresponding classification result. Yet, we have to note that single-row data insertion produces a lot of communication overhead and redo log writes to ensure durability and consistency (see [463]), which causes similar performance problems as discussed in Section 10.2.4. Thus, the stream is transformed to FV chunks of size 5000 and each of these chunks is forwarded to TDB instead. This way, significantly fewer transactions are issued and so RAC can operate in its preferred processing mode. Both final operations (see alerting and insertion) correspond to sinks in Flink terminology whereas the batch transformation is the preceding operation of the insertion. An overview of these actions in given through Figure 10.10.

10.2.7 Concluding Picture

The previous sections of the conducted system deployment focused on each architectural component. The purpose of this Section 10.2.7 is to supply a global picture where all those pieces are put together showcasing the concluding HFIDS.

Let us start with the data ingestion. We expect bidirectional IPFIX data to arrive from one or multiple flow exporters that in turn are fed by IP packets from observation points strategically situated in the network landscape to monitor. Beside network-related data, an option accepted by our system is the incorporation of host events as well, which can be collected on individual computer systems enriching the context of flows. Both data

[7] `https://suricata-ids.org`

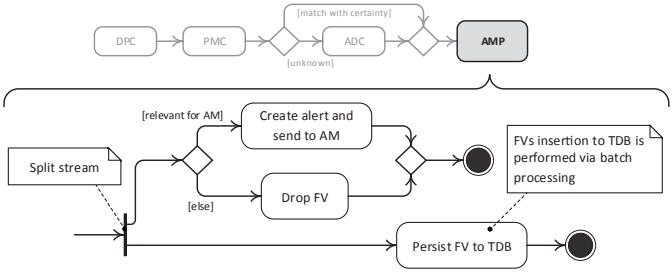

FIGURE 10.10: AMP processing where the stream of FV is split to alert suspicious FVs and perist all FVs at TDB concurrently

sources are buffered intermediately using Kafka as message queue and organized in two separated logical containers, which are topics in Kafka jargon. From there, they have to pass through MIP. This pipeline is primarily realized by the stream processor Flink that is the only subscriber of both topics consuming Kafka messages in a stream-like fashion. Within Flink, data undergo a preprocessing step DPC creating exactly one FV for one arriving flow. During preprocessing, involved treatments depend on the mode the HFIDS is running. We distinguish by four possible options determined by the FFC, i.e. the selected feature combination of interest. Combination F00 is the most basic form where the HFIDS is expected to assess flow data only. On the other side, F0W also extracts traffic statistics by performing a neighborhood analysis among flows using dedicated window computations provided by Flink. Both FFCs can also be enhanced with the aforementioned host data resulting in the two new combinations FH0 and FHW. Yet, this enhancement requires the fusion of the FV stream and the host event stream employing a global window acting as temporary buffer. If one or multiple host events can be correlated to an existing FV using that buffer, the fusion can be considered successful. In several situations, however, no host data can be associated and so the FV is sent to the final activity of the DPC after some waiting time without successful correlation. The same is the case for FVs successfully correlated. This last activity of DPC takes care to normalize and discretized specific features of each FV. After the preprocessing period, FVs are forwarded to PMC where the system tries to find one or multiple patterns for each FV in its underlying pattern repository which we refer to as local PDB. It is implemented using an in-memory cache solution, i.e. TT cache. In that sense, a pattern corresponds to a decision rule having the practical benefit to represent a piece of transparent knowledge that can be directly applied to classify the FV under inspection. Once matched, it explains the prediction and operational staff is supported to initiate immediate follow-up activities on potential security incidents. Dependent on the matching, two basic results have to be distinguished. The first result type rates a FV as “unknown” because either no pattern matched at all or a tie occurred, i.e. PMC is uncertain about a decision. The second result type comprises a definitive decision. Either a single pattern fired exclusively or there is consensus among multiple matching patterns evaluating the FV to be “malicious”, “benign” or “anomalous”. Note that in case of any malicious pattern match, an alert is raised by the system anyway. Right after PMC, an optional inspection is conducted by the anomaly detector ADC. It sets

in if the pattern matching is uncertain applying its underlying ensemble of black box ML models. A malicious classification result at this stage also leads to an alert. Passing both detection points, classified FVs are prepared to leave MIP, which is comprised by the AMP activity. This final activity within Flink persists FVs at the traffic repository TDB hosted by the DB system RAC. Concurrently to this action, alarms are rendered as well and redirected to the alert management unit, i.e. AM. Once FVs are persisted, TDB fulfills two duties. On the one hand, it serves as input for PBC, which generates new patterns or updates existing ones utilizing advanced in-DB analytics inside RAC that are maintained in the global pattern repository PDB. From there, patterns are replicated to local PDBs in an incremental manner to loop back acquired knowledge to MIP. On the other hand, TDB serves adaption at ADC. This is implemented by its offline component that periodically updates the online ensemble hosted by Flink in case its predictive performance degrades. If the data distribution changes due to dynamics in the underlying network landscape to safeguard, a clustering algorithm anticipates such situations and relabels earlier misjudgments in TDB as "anomalous". This way, both ADC and PBC can adjust properly to such situations. Yet, this class is just a meta class, which needs supervision occasionally in order to provide the true class label. This process is initiated by AM. Resolved anomalous clusters are sent back to the offline component of ADC that publishes this gathered GT to TDB from where both the ensemble and corresponding patterns can be updated accordingly. A comprehensive picture of this proposed HFIDS is outlined in Figure 10.11. Note, we ignored to depict an additional service that keeps both data producers and consumers of Kafka synchronized, which is realized by Apache ZooKeeper[8] running on a dedicated machine with minor resource demands.

Given this recap of the implemented HFIDS architecture, we also want to point out scalability properties of involved subsystems. In this respect, we can distinguish between five main components. First, there is the message queue Kafka taking care of raw data ingestion, which is technically a cluster of computer nodes carrying portions of each topic in a distributed manner. By appending more nodes to the cluster, workload can be spread more evenly delivering scale-out functionality. The consumer of Kafka messages is Flink driving MIP. It is also a cluster that is horizontally scalable. By adding more Flink nodes, more computing power is applicable and more operator subtasks can run in parallel. In fact, large portions of the processing can be realized through operator chains where a repartitioning of the data stream is avoidable (see gray arrow annotations in MIP of Figure 10.11). Yet, this is highly dependent on the selected FFC requiring further practical assessment to which degree throughput rates can be increased. RAC also attempts to reach scalability. It can be obtained by appending more nodes that share a common disk-based DB. As such, the load from Flink can be balanced appropriately while supplying distributed query processing for PBC at the same time. The only two components in our current implementation that are not horizontally scalable are the offline component of ADC and AM. Nevertheless, an attempt can be made to scale them vertically. These mentioned scalability attributes per subsystem are symbolized in Figure 10.11 either by a single computer (see scale-up) or a group of computers (see scale-out) both colored in gray.

[8] `https://zookeeper.apache.org`

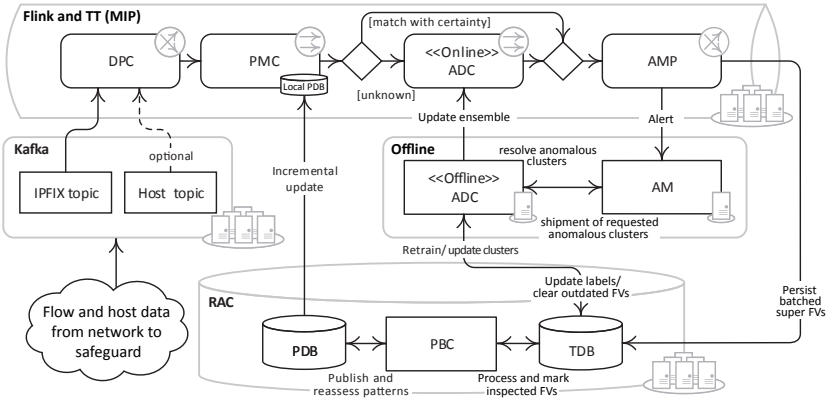

FIGURE 10.11: Implementation and deployment of the proposed HFIDS as a computing cluster with Flink as main management pipeline to process the arriving stream of flows and host data buffered through the message queue Kafka that are passed to RAC once inspected for both the extraction of patterns using in-DB analytics and serving as repository to update ADC; scale-out potentials can be obtained by allocating more nodes to each subcluster (Kafka, Flink and RAC) while the offline components (ADC (offline) and AM) can only be scaled up

10.3 System Evaluation

Given the described deployment of the HFIDS, let us evaluate the system in practice with all of its assembled components eventually. Therefore, we outline the hardware environment that serves as workbench for all of our experiments alongside with basic data sets and employed methods as well (Section 10.3.1). Based on that, operating points are figured out for the main subsystems (Section 10.3.2) and utilized to assess the HFIDS in simulated drift scenarios (Section 10.3.3). We further consolidate outcomes applying the system on a much larger network capture and contrast the classification performance of our system with the results of other stream mining algorithms (Section 10.3.4). In addition, the scalability potentials are highlighted by stress testing the overall system (Section 10.3.5). Lastly, we provide a discussion on gathered findings during the evaluation (Section 10.3.6).

10.3.1 Experimental Setup

During the course of deploying the HFIDS, we realized that the system in charge has to pursue a set of diverse technical objectives in order to monitor and protect a given network infrastructure. Among others, these include the efficient buffering and online processing of arriving data, their persistence and retrieval respectively which in turn forms a distributed system or cluster of interconnected subsystems. Beyond that, most of those subclusters are designed to be elastic such that they can be extended horizontally to meet expected data throughput rates. Thus, it is not difficult to acknowledge that

VM type	vCPUs	RAM	HDD	Software configuration
Flink	8	20	20	Ubuntu 14.04.5 (64 bit), Apache Flink 1.4.2 (Scala 2.11), Oracle TT 11.2.2.8 (64 bit)
Kafka	4	16	40	Ubuntu 14.04.5 (64 bit), Apache Kafka 0.10.0 (Scala 2.11)
Producer	8	20	40	Ubuntu 14.04.5 (64 bit)
RAC	8	24	70	Microsoft Windows Server 6.3 (std., build 9600), Oracle DB 12c (enterprise, 12.1.0.2) as RAC (64 bit)
ZooKeeper	2	4	40	Ubuntu 14.04.5 (64 bit), Apache ZooKeeper 3.4.9
ADC (offline)	12	16	20	Ubuntu 14.04.5 (64 bit)
AM	4	8	20	Ubuntu 14.04.5 (64 bit)

TABLE 10.1: Hardware and software profile of individual cluster nodes; RAM and system hard disk drive capacity (HDD) are presented in Gbytes

a large body of hardware resources has to be bound to run all system components properly. To allocate these resources, we are not issuing bare-metal computers but a very sophisticated virtualization that is hosted inside our campus facilities. In contrast to earlier experiments in this work where we already relied on virtual testbeds (see Chapter 6, 7 and 9), the underlying hardware of this provisioned virtualization platform[9] is much more competitive and comparable to commercial virtualization services. At this point, we want to emphasize that parts of this infrastructure were already utilized in other case studies to assess energy efficiency factors in private cloud systems (see [476, 477]). Hence, we refer to these mentioned references offering further technical details concerning the system's layout and dimensioning. In terms of hardware allocation, a resource contingent could be reserved to build up our test landscape on top of this infrastructure. As such, a set of VMs was assembled to comply with each subsystem in our hybrid intrusion detection cluster. An outline of each individual VM type along with tailored hardware and software profiles is presented in Table 10.1 serving as basic entities during the experiments. In that sense, the producer VM handles a special task. It simulates the data generation that would normally be provided via one or multiple mirror ports inside the network site to safeguard. This also includes the flow exporting as well as the partial delivery of host information constituting the input for the HFIDS. All other VM types correspond to their conventional tasks.

Apart from the virtual workbench, further importance has to be attached to the benchmark data our experiments are based on. In this regards, two aspects are vital to assess the developed HFIDS. On the one hand, we have to deal with nonstationary environments, which essentially means that utilized data should comprise concept drift characteristics. This point is very critical not least because little is known about drifts in the context of practical network security (see Section 5.2) and so very few evidence about such properties is provided in existing network captures. We are only aware of one data set that is often proclaimed to contain concept drift and used in literature, i.e. KDD-Cup 99 (e.g. [102, 478, 479]). Yet, we already mentioned its issues with re-

[9] VMWare vSphere 6.5.0 and Mirantis OpenStack 9.2 for Mitaka on 4× Dell RS620 (2× Intel Xeon CPU E5-2650v2 (2.60 GHz), 256 Gbyte RAM, 2× 300 Gbyte SAS HDDs, 2× 1 Tbyte SSDs) interconnected with 1× Arista 7150S and 1× Arista 7050S via 10 Gbit/s ethernet. Additionally, this server cluster is attached to a disk array consisting of 2× NetApp E2724 shelves holding 24× 900 Gbyte SAS HDDs configured as RAID-5 via 16 Gbit/s fiber channel, which carries system and application data of existing VMs in vSphere.

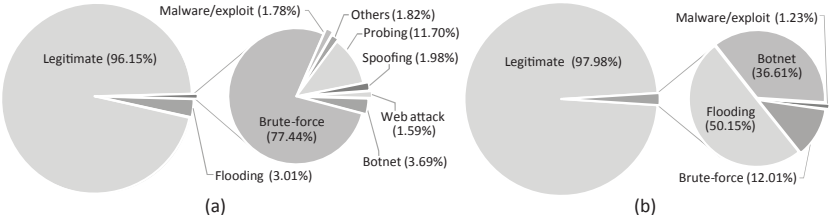

FIGURE 10.12: Attack distribution breakdown for (a) DS-I and (b) DS-II

spect to age and design flaws (see Section 6.2) such that its usage herein is prohibitive. Even though these points would be neglected, we are unable to employ KDD-Cup 99 because its main entities are connection-based in lieu of flows (see Section 5.3) and so many desired features elaborated in Chapter 7 are not evaluable. On the other hand, a challenge of the HFIDS is to handle class imbalance where attacks are far less observed than legitimate data (see Section 5.5 or 8.1). In order to counter these major points, we make use of the basic data sources already employed in Chapter 7, i.e. DS-1 (a mixture of NDSec-1 and legitimate enterprise data) and DS-2 (a combination of the ISCX-2012 and CTU-13 traces). On this occasion, however, we put emphasis on creating skewed class distributions. Another point in this regard is the rather poor GT of ISCX-2012 and CTU-13. Therefore, we reassessed these captures manually to obtain more meaningful class labels and synchronized them with the attack categories of NDSec-1 (see Section 6.4.1 and Table 6.2 in particular). As a result of this additional effort, we are not just capable to analyze performances within a binary classification setting but also can present results setting up multiclass scenarios. The outcome of these actions is depicted in Figure 10.12 showcasing the data distribution of those new data collections. By analogy to the two data sources of Chapter 7, we call them "DS-I" and "DS-II" respectively. The problem, though, is that both are still not incorporating concept drift. Hence, we make use of a methodology provided by the MOA framework that introduces synthetic drifts by fusing data streams drawn from different distributions (e.g. [112, 426]). In essence, it builds up on the "sigmoid function"[10] given by

$$f(t) = \frac{1}{1 + e^{-4(t-t_0)/w}} \,. \tag{10.1}$$

This customizable function $f(t)$ can be understood as the probability to select data from a specific data source controlled by the central change point t_0 and the change period w. Its characteristic is concretely pictured in Figure 10.13 (a) where choosing data from the source is very unlikely at the beginning ($t < t_0 - w/2$). As we approach the change point, the probability grows until all data are finally selected from the source ($t > t_0 + w/2$). Thus, we can combine two of these sigmoid functions to model a transition from one distribution to another at t_0 either by a smooth shift (w is chosen to be large) or by a rather swiftly one (w is chosen to be small). This idea can be formalized by the following operation

$$s := a \oplus_{t_0}^{w} b \tag{10.2}$$

[10] This function is a generalized variant of *sgn* that we already used in (8.2).

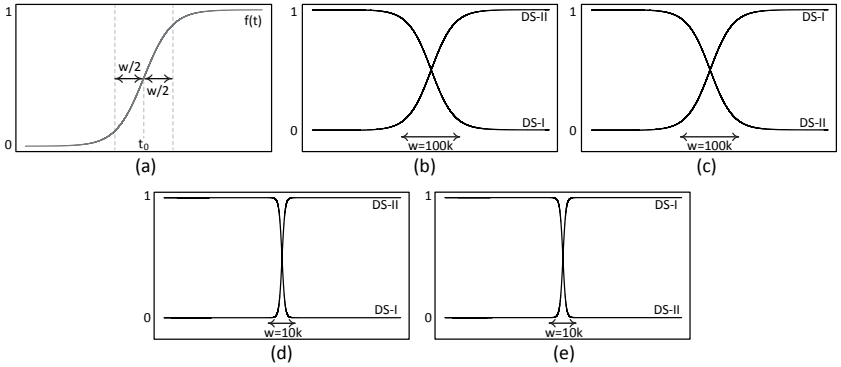

FIGURE 10.13: Drift simulation approach: (a) illustration of the sigmoid function and a schematic overview of the four different transition scenarios from DS-I to DS-II and vise versa depicting (b-c) gradual and (d-e) abrupt changes in the overall data distribution

that joins two data streams a and b sequentially according to t_0 and w towards a new stream s with the probabilities

$$P(s(t) = a(t)) = e^{-4(t-t_0)/w} \cdot f(t) \tag{10.3}$$

and

$$P(s(t) = b(t)) = f(t) \; . \tag{10.4}$$

In our case, we use this methodology to render four different data sets with synthetic drifts. They are outlined schematically in Figure 10.13 (b-e) with the central position of change $t_0 = 3 \cdot 10^5$. In the first two variants, we model a gradual transition from DS-I to DS-II and vise versa with a length $w = 10^5$ whereas in the last two variants a rather abrupt transition between the two data sets is established using a width $w = 10000$. To reference these four assembled sources, we name them "DS-I-to-II-100k", "DS-II-to-I-100k", "DS-I-to-II-10k" and "DS-II-to-I-10k" in consecutive order. Verifying the desired drift characteristics, we applied two drift detectors as given in [367, 368] right before our experiments actually commenced. Both confirmed at least one change in the data distribution. Yet, the detection varied slightly per detector and data set but still occurred within $t_0 \pm w/2$. Additionally, we would like to state that assessments for incremental drift scenario are neglected at this point because they are very specific and rather difficult to model in a general setup considering higher dimensionality. Please also note that a qualitative test on correlating flow and host information (see combination FH0 and FHW) cannot be considered either. Even though DS-I is equipped with such supplemental data, DS-II is not providing appropriate host data at all, which, combined, conflicts with the fusion process of both data sets introduced just recently (see (10.2)). Anyhow, a quantitative assessment regarding the combination of flow and host data is conducted when we analyze the system's scaling potentials. To further extend our evaluation, we incorporate a new network trace in addition that reflects real enterprise traffic which is beyond synthetic scenarios. It was tapped at the backbone infrastructure of our research partner EDAG at the end of 2017. Using this backbone, we were able to

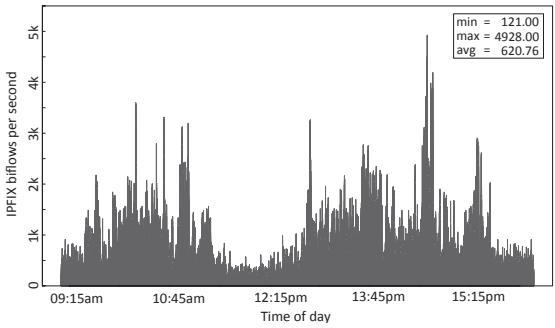

FIGURE 10.14: Flow distribution of the captured enterprise traffic over time

collect network communication between multiple departments on different floors across two office buildings whereas the line speed at the extraction point was designed for throughput rates of up to ten Gbit/s. Consequently, we could not only sense a variety of traffic but also monitor realistic workload situations during different day periods catering for a total traffic volume of 160 Gbytes with sporadic peaks of 6.50 Gbit/s. An overview of this real-world enterprise capture is outlined in Figure 10.14 and Figure 10.15 (a) showcasing the flow traffic distribution over time and the four most dominant network services respectively. As this data collection is attack-free verified by applying the latest version of the IDS Snort, additional modifications had to be considered in order to turn this capture into a supplemental benchmark data set. Therefore, we reused flows of NDSec-1 flagged as malicious to salt the enterprise traffic by using a similar strategy as for DS-I. As a result, the new compilation had a very high competitive imbalance ratio and reflected network content of almost a complete working day with roughly 9.5 million biflow records in total. Moreover, it is interesting to figure out whether this new data set comprises drift characteristics out-of-the-box. Our analysis in this respect is two-fold. Several distribution changes were notable employing the change detector proposed in [368] particularly when considering flow data exclusively. Incorporating traffic statistics, only minor changes could be disclosed using this approach. Yet, these findings could be neither confirmed for F00 nor for F0W utilizing the drift spotting algorithm in [367]. From this perspective, we cannot rule out drifting characteristics in the general case for this data set but even if they are included, we remain uncertain in terms of their duration and type. Out of simplicity, we name this very representative data source "DS-III" whose attack distribution is outlined in Figure 10.15 (b). Note that due to EDAG's privacy regularizations, we cannot make public these real-world network traces.

Given these isolated components of our experimental setup, let us briefly discuss how we integrate them to assess the HFIDS systematically. A major problem in this context is the high amount of variables to configure and to preset for both ADC and PBC in practice. On these grounds, we devote the first part of our analysis to this matter and determine proper operating points by incorporating our synthetic drift scenarios exclusively. Two reasons of choosing these data sources are noteworthy. On the one hand, the appropriate adjustment of learning algorithms ideally requires a profound data

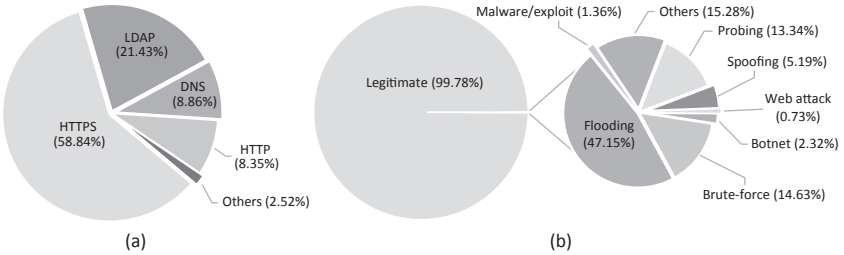

FIGURE 10.15: Characteristics of the enterprise capture with (a) the four most dominant network services and (b) the traffic distribution after being salted with attacks from NDSec-1 ready to be utilized as benchmark data set DS-III

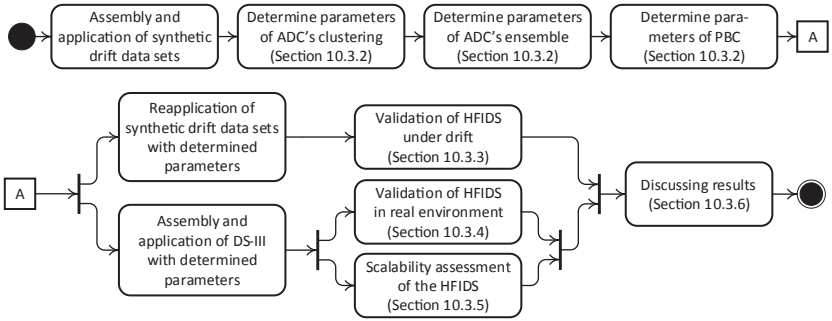

FIGURE 10.16: Organization of our experiments to assesss the proposed HFIDS in practice depicted as activity diagram

basis with sufficient variability in order to avoid overfitting aspects. This is provided by these four data sets per FFC all carrying their own particularities. On the other hand, we are interested to obtain solid parameters to handle drift characteristics, which is also comprised by these benchmark sets to a certain degree pointing at the gradual and sudden change scenarios involved. To proceed with these sources, we work out the best parameter values for the subsystems ADC and PBC. We start by assessing ADC in isolation. The shielded assessment on this component has a simple line of thought. There are worst case scenarios where PDB might be empty and so the entire classification task has to be lifted by ADC alone. This setting is likely to appear when running the HFIDS from scratch where ADC is already trained but PBC is not involved in the initial training process. Based on these outcomes running ADC separately, we turn over to examine operating points for PBC, which directly depends on the output of ADC in the most extreme case. As such, we gain the final parameter lineup for further investigations. These include the complete examination of the HFIDS in terms of drifts, real environments and scalability potentials right before we discuss results. A graphical presentation of this strategy to analyze the HFIDS from different perspectives is depicted in Figure 10.16.

10.3.2 Identifying Operating Points

Throughout the developing work, several parameters were introduced to guide the learning process of the HFIDS. These include the cluster sizing to identify new structures in the data, confidence thresholds to train on newly evolving data or establishing an appropriate rule aging cycle just to name some important aspects. Hence, finding reasonable values for these variables is an essential step right before the system can actually deliver desired protection for the network landscape of interest. In this section, we work towards such operating points by applying both knowledge gathered over the course of the development combined with dedicated test runs on all four assembled drift data sets on either subsystem. Note that, we separate the analysis into two distinct groups in order to evaluate each subsystem per FFC, which can be justified along the different features and the number of dimensions comprised.

According to Figure 10.16, we lift off this identification process by assessing ADC first. As its clustering operates independently from the ensemble algorithm, beginning with the examination of DBStream is free of care. Hence, a proper metric has to be employed that provides a comprehensive view to the clustering quality. Considering the years of intensive clustering research, several indicators have been manifested in literature (e.g. [480, 481, 482]). Herein, we partly make use of the prominent entropy-based "V-measure" [481] that incorporates the notion of "completeness" and "homogeneity" at the same time. In this respect, a cluster result is said to be complete if all data points or examples from one class fall into one dedicated cluster. The homogeneity score in turn is less restrictive and puts emphasis to the purity of the clustering similar to the purity of a decision rule (see Definition 9.4). If this score equals to one, it signals that each produced cluster only summarizes examples from a single class. The clustering is said to be homogeneous, while clusters containing data points from different classes are inhomogeneous and, thus, undesirable resulting in a low value close or equal to zero. For our purpose, the latter concept of the V-measure is just sufficient to evaluate the clustering algorithms because we are eager to identify new quality clusters once evolving from the data stream in order to accurately assigned them to the anomalous class as a first step. In the second stage, our goal is to facilitate operations at AM where the delivery of homogeneous clusters for the labeling process is more important than the completeness property. Recalling that DBStream basically relies on one input parameter, i.e. the micro-clustering radius, we present its average homogeneity score[11] over the radius on all four synthetic drift data sets per FFC in Figure 10.17. In essence, the results show that with increasing radius we also obtain more homogeneity up to a certain point. Once surpassed, a degradation of the overall homogeneity is noticeable. While the best radius obviously is obtained at that particular turning point, the downside of our evaluation is its fluctuation with respect to all benchmarks such that no global radius can be exposed right away. Therefore, we refine the analysis by nominating the radius as the best where the pairwise homogeneity scores among the benchmarks are maximized with minimal variance. This yields a combined radius of 0.16 for F00 and 0.04 for F0W that still produces a suitable cluster quality on all data sets given our trade-off analysis.

[11] Note that due to our data stream setup, we used a sampling interval of 30000 FVs that is averaged to obtain final scores.

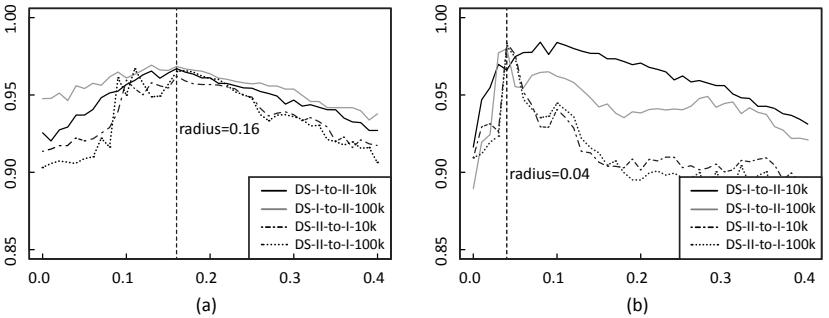

FIGURE 10.17: Change of the cluster homogeneity score over the micro-clustering radius on all four simulated drift scenarios per combination (a) F00 and (b) F0W where the x-axis shows the radius and the y-axis highlights the average homogeneity; dashed vertical lines indicate the operating point with the best combined outcome

We move over to the ensemble encompassed by ADC and assess its two input parameters w_{min} and ψ_{min} while incorporating the clustering concurrently. Yet, before we proceed with the optimization of both variables, let us briefly review and discuss their extremes. Beginning with w_{min}, its basic intention is to adhere a proper classification performance among the individual ensemble members not falling below this particular threshold. Hence, it controls whether each member is compliant with most recent flow data or engages an individual update otherwise. Obviously, a value close to 100% is tempting but such a setting indirectly has an impact to the retraining frequency in addition, which may pose higher demands from a computational perspective. Both, the opportunity for better predictions and requiring additional resources are opposing eventually and a heavy model fluctuation over time is certainly not contributing much to a transparent decision-making process if this aspect can be attributed to ADC at all. On the contrary, low values permit a relaxation of these mentioned affairs but also increase the risk of potential misclassifications and wrong assumptions during future adjustments. Setting up the other parameter ψ_{min} is not less critical. It determines whether to consider classified flows in TDB for retraining purposes and, thus, has a direct impact on ADC's adaptivity. A high value near 100% seems to be the preferred choice because we want to be as confident as possible about the predicted class labels of flows before taking them into account for upcoming training rounds. This opportunistic view is misleading though. On the one hand, it is somewhat contradictory to the postulated diversity an ensemble should have in order to be effective. Such a group of different experts is unlikely to reach a constantly high agreement along a prediction and so very few cases are consider after all. This way, the ability to take advantage of classified flows as new input for retrainings vanishes over time. On the other hand, a high consensus among the members is neither an ideal setup. The adaptivity of ADC's ensemble would be restrained as it only can retrain on data it already predicted correctly. This limits the chances to enhance existing knowledge beyond that point. Moving ψ_{min} to the other extreme resolves the mentioned issues but also increase the risk of considering flow data that are falsely classified, which complicates a proper adaption process at the other end due to the noisy pseudo GT just produced. Overall, those discussed points pose

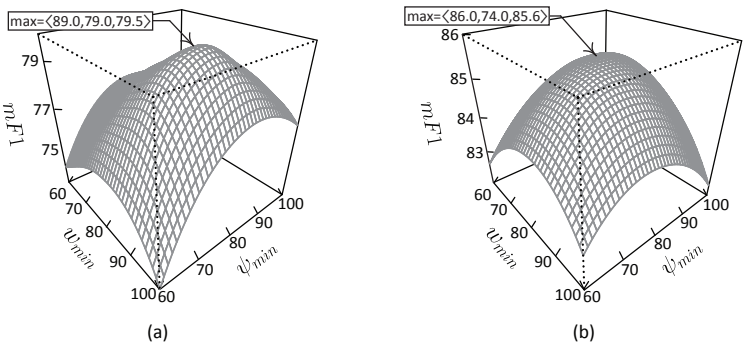

FIGURE 10.18: Interpolated grid search result on all four simulated drift scenarios showcasing the average $mF1$ as w_{min} and ψ_{min} are changed for ADC's ensemble component aggregated by combination (a) F00 and (b) F0W; all axes are expressed in percent

a dilemma and require a profound quantification of both parameters. Therefore, we analyze their mutual characteristics in different situations performing a conventional "grid search" with the $mF1$-score as predefined objective function to maximize enclosed in a binary classification problem, i.e. the task to find an appropriate separation between benign and malicious FVs. It is important to state that the anomalous class is not considered at this point due to its meta class characteristic. As such, the underlying GT can never match FVs classified as anomalies refusing the calculation of hit rates for that particular class. Yet, this class is included as part of the pseudo GT to determine internal model performances of ADC's supervised component attempting to solve a three-class problem instead. Within this setting, w_{min}-values actually correspond to the three-class $mF1$-score, which is different to the utilized error rate in Section 8.3. The outcome of this evaluation is depicted as interpolation in Figure 10.18 showing the average $mF1$ behavior as we change w_{min} and ψ_{min} aggregated per FFC where the first 30000 FVs with GT are used to initialize ADC constituting an EVL setup (see Section 10.2.5). These results state that even when setting w_{min} and ψ_{min} to a high value, a solid $mF1$ is not reached necessarily which we just discussed. Indeed, the maximum $mF1$ is obtained if a discriminating balance of both is taken in the range between 60% and 100%, i.e. $w_{min} = 89.00\%$, $\psi_{min} = 79.00\%$ for F00 and $w_{min} = 86.00\%$, $\psi_{min} = 74.00\%$ for F0W justifying our operating points per FFC. Note that, the intermediate grid search results per benchmark data set are highlighted in Appendix A.5. In this experiment, we predefined the initial sizing factor $\alpha_{\perp} = 1$ and potential retrainings to be triggered every 30000 flows.

Based on the operating points identified for ADC, let us prepare the scene to optimize the pattern creation and matching respectively. In this regard, we are not only interested in sheer predictive performance aspects but also eager to ensure a reasonable participation in the classification process recalling PBC's abstaining property (see Section 9.3.3). Hence, an appropriate objective function has to be defined addressing both points. Apparently, we look for a performance metric that maximizes t-$mF1$ and minimizes ABR at the same time. To facilitate this demand, we can turn both into a single

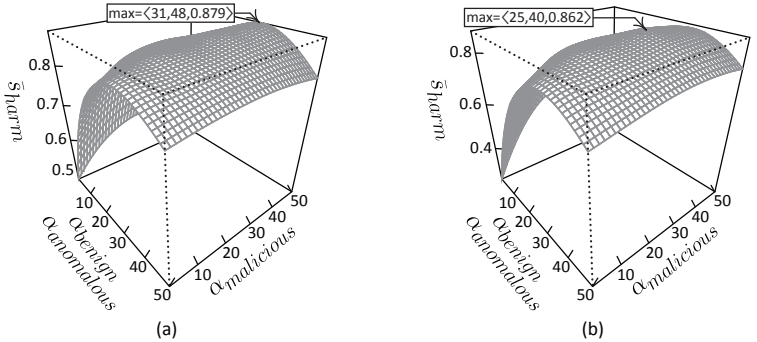

FIGURE 10.19: Interpolated grid search result on all four simulated drift scenarios showcasing the average $\bar{s}_{harm}$ as the pattern aging α_c is changed aggregated by FFC: (a) F00 and (b) F0W

maximization problem, which can be accomplished by the introduction of

$$\bar{s}_{harm} = 2 \cdot \frac{t\text{-}mF1 \cdot (1 - ABR)}{t\text{-}mF1 + (1 - ABR)} \,. \tag{10.5}$$

This new measure $\bar{s}_{harm}$ represents the harmonic mean of PBC's tentative $mF1$-score and the complement of the abstaining characteristic, which reflects the proportion of certain classifications taken by PMC in relation to all prediction attempts made by the HFIDS. Besides this measure, we cannot, however, take the same grid search method as done in the previous paragraph to tune ADC. The amount of variables to be tweaked for PBC is simply to high imposing a rather impractical task from a computational standpoint even when considering parallel processing. Hence, we proceed incrementally by performing a two-step grid search optimization: we seek for an adequate rule aging cycle first, i.e. setting up α_c, which implies that other variables have to be preset. This is not an issue at all as the tolerated error $\beta = 0$ and the leap constant $t = 1$ can be safely set as default configuration. Once, a solid rule aging is found, we fine-tune β and t accordingly in a second step. Note that the capacity of the partial memory w_κ, batch size $|T|$ and pattern simplification function $g(j)$ are excluded from further tweaking. Instead, we fix these variables reasonably based on our gathered experience and keep this fixation throughout the remaining experiments, i.e. $w_\kappa = 5000, |T| = 1000$ for either FFC and $g(j) = 2/5 \cdot e^{-j/4}$ if we consider native flow data only or $g(j) = 3/7 \cdot e^{-9j/32}$ if flows are combined with traffic statistics. Obtained results are pictured in Figure 10.19 and 10.20 where we use the outcome of ADC with its tuned parameters as input for PBC. Additionally, we assume that produced patterns are directly available at PMC. For both FFCs, the aggregated outputs reveal that with increasing age α_{benign} and $\alpha_{anomalous}$ respectable performances can be gained quickly. Within the range $19 \leq \alpha_{benign} \leq 47$, we reach $0.80 \leq \bar{s}_{harm} < 0.88$ neglecting $\alpha_{malicious}$ for a moment. Ages beyond this range lead to a constant decrease of $\bar{s}_{harm}$ and, thus, to unfavorable outcomes. If we also take into account $\alpha_{malicious}$ inside the mentioned corridor and increment it, it is observable that adequate performance scores are reached very lately. In effect, the best $\bar{s}_{harm}$ is attained when setting $\alpha_{benign} = \alpha_{anomalous} = 31, \alpha_{malicious} = 48$ for F00

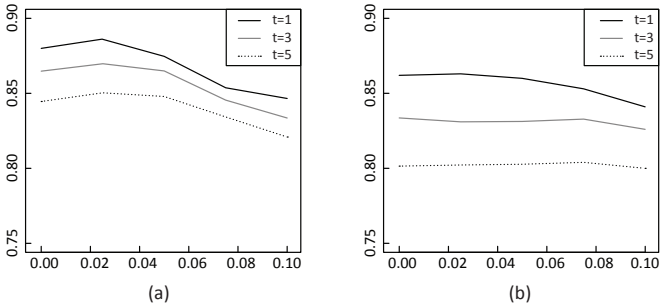

FIGURE 10.20: Change of $\bar{s}_{harm}$ over the permitted error β and leap constant t on all four simulated drift scenarios aggregated by FFC: (a) F00 and (b) F0W; x-axis shows the parameter β and y-axis highlights the average $\bar{s}_{harm}$

and $\alpha_{benign} = \alpha_{anomalous} = 25, \alpha_{malicious} = 40$ for F0W. This result signifies that PBC benefits if $\alpha_{malicious}$ is chosen to be proportionally larger than α_{benign} and $\alpha_{anomalous}$ by a factor of at least 1.50. Note that intermediate results per drift data set are illustrated in Appendix A.5. Taking these results as baseline, let us briefly report about tweaking β and t. In this respect, changing β from 0.000 to 0.025 improves $\bar{s}_{harm}$ further to a score of 0.886 considering $t = 1$ using combination F00. Taking into account higher values either for β or t only degrade PBC's performance. Regarding F0W, no clear improvement is notable. Changing β and increasing t in this context only leads to a deterioration of $\bar{s}_{harm}$ in our experimental setup.

A wrap-up of all identified operating points is composed in Table 10.2 per subsystem and FFC. Given that parameters for FH0 and FHW could not be obtained due to a lack of available host information in some of our considered data sets (see Section 10.3.1), we expect that settings for those missing FFCs still comply to their respective base FFC. This means that parameters for FH0 are presumably similar to F00 and FHW relates to F0W likewise. An exception to this premise might be the cluster radius of ADC. As opposed to other parameters, it turned out to be more delicate in our experiments. Thus, it is very plausible that this specific parameter requires further adjustment as dimensionality is increased.

10.3.3 Assessment on Synthetic Drifts

With the operating points identified in the previous Section 10.3.2, we reexamine the synthetic drift scenarios herein. This time, however, we apply the entire HFIDS with this promising configuration instead of making isolated assessments. Therefore, we utilize the same warm-up phase where the first 30000 FVs are employed to train the system and PDB is considered to be empty initially. In addition, we expand our analysis to multi-class classifications, which can be considered significantly more challenging particularly under class imbalance. Hence, the macro-averaged F1-score $mF1$ is used to measure the classification performance across the nine classes "benign", "botnet", "brute-force", "malware/exploit", "flooding", "probing", "spoofing", "web attack" and "others" (see

System	Parameter	F00	F0W
ADC	cluster radius	0.160	0.040
	w_{min}	0.890	0.860
	ψ_{min}	0.790	0.740
	$\alpha_{\perp}$	1	1
	train/test ratio	4 : 1	4 : 1
	retrain interval	every 30k FVs	every 30k FVs
PBC	α_{benign}	31	25
	$\alpha_{anomalous}$	31	25
	$\alpha_{malicious}$	48	40
	t	1	1
	$g(j)$	$2/5 \cdot e^{-j/4}$	$3/7 \cdot e^{-9j/32}$
	β	0.025	0.000
	$\lvert T \rvert$	1000	1000
	w_{κ}	5000	5000

TABLE 10.2: Quantified HFIDS parameters under different drift scenarios

GT categorization of NDSec-1). Yet, to be consistent with the obtained operating points, we abstract the internal performance measurements of ADC's ensemble to a three-class problem and also state binary performances indicated by "$mF1_b$", which corresponds to $mF1$ in the previous section. To meter the number of detected anomalous FVs by the HFIDS over the course of the benchmark data, a further measure is introduced which we name "anomaly detection rate" (ADR). It simply counts identified anomalies and relates this number to all classifications performed in a specific range. This way, ADR combines two properties. On the one hand, it expresses the degree of change comprised in the data with respect to new phenomena our HFIDS is able to expose but unable to classify in one of the nine classes, i.e. a kind of uncertainty (see Section 8.3.2). On the other hand, ADR provides an intuition of the data proportions, which are not covered by $mF1$ and $mF1_b$ recalling that detected anomalies correspond to a meta class during the absence of their true class. To address the latter issue, we incorporate AM to resolve anomalous clusters with their real GT, i.e. an active learning mechanism (see resolution process in Section 10.2.5) that was omitted so far. The motivation comes from the severity of the synthetic drift scenarios producing high ADRs and a moderate performance to predict other classes correctly in the drift phase and afterwards. The HFIDS cannot fully recover from such situations alone without being guided by at least a small number of true class labels (see cumulated result of best operating points in Figure 10.18). To simulate AM's resolution during our experiments in an automated fashion, we draw on a straightforward approach: besides cluster centers and radiuses, a maximum proportion of 5000 FVs comprised by each cluster is stored in addition, which are used to identify a cluster's class label. If the cluster is homogeneous, i.e. all included FVs in the cluster share the same label, we output that corresponding class. In case of a inhomogeneous cluster, a majority vote is applied to obtain the cluster label in analogy to the mechanisms introduced for the ensemble H and PMC. Note that AM can initiate the resolution process at any time but obviously the ideal time to start is right after the drift. Hence, we involve our automated version of AM at data point $3.60 \cdot 10^5$ for the first time, which is applicable to all four drift data sets because $3.60 \cdot 10^5 > t_0 + w/2$. From this point onwards, the resolution is triggered every 60000 FVs as long as we regain similar performances than attained before the drift. In some situations, however, we experience that this undertaking cannot always be reached.

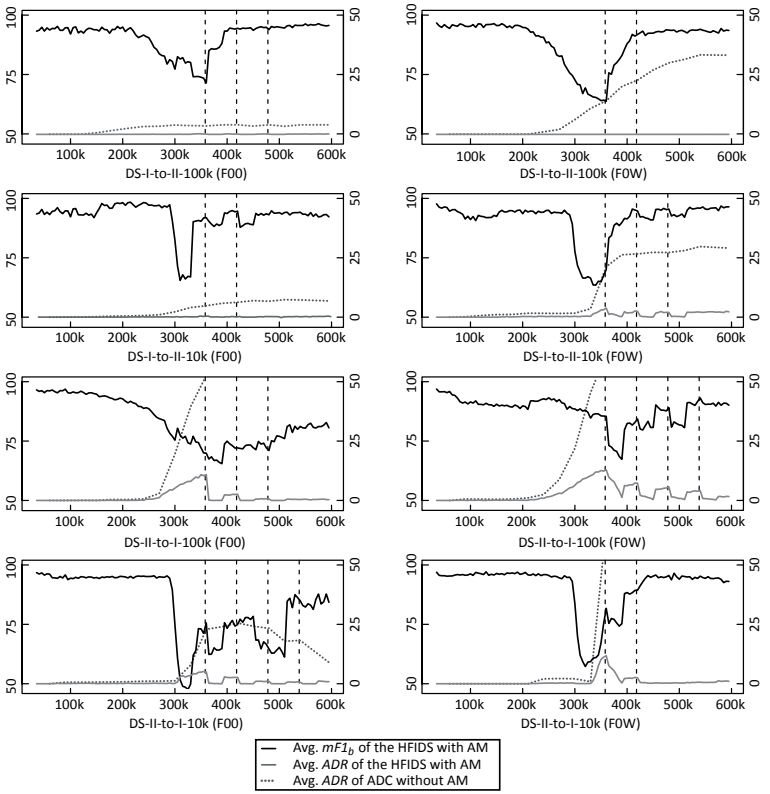

FIGURE 10.21: Average predictive performance of the HFIDS using $mF1_b$ and ADR over the course of all four synthetic drift data sets per FFC; dashed vertical lines indicate moments where the AM resolution process sets in; first and second y-axis show the scales for $mF1_b$ and ADR respectively both in percent

Given these prerequisites, we move over to the experimental results applying the HFIDS under synthetic drift. They are visualized in Figure 10.21. On all benchmark data, the HFIDS obtains a quite stable classification performance right after the warm-up. Such a behavior is not symptomatic necessarily, which can be justified along both the evolving nature of streams and the incremental learning principles applied. Nevertheless, the acquired outcome demonstrates that initial training data basically reflect data shapes of the live phase and that the system can anticipate minor structures through internal adaptions at ADC and generated rules at PBC that were not seen during training. This is supported by produced F1-scores where we reach an average 94.29% for $mF1_b$ on all benchmark data sets measured over the first $2.50 \cdot 10^5$ FVs. From this point on, the classification performance starts to degrade due to concept drifts whose severity depend upon either drift type. While in both drift settings, similar average F1-scores are achievable within the FV range $2.50 \cdot 10^5$ and $3.50 \cdot 10^5$ ($mF1_b = 80.93\%$ for the gradual

drift and $mF1_b = 80.20\%$ for the abrupt one), the degradation is more radical during the abrupt change. Performances drop by 30% and more all of a sudden, which can neither be compensated by the introduced meta class nor by internal adaption mechanisms. After those drifts, i.e. the FV interval $3.50{\cdot}10^5$ to $6.00{\cdot}10^5$, the classification performance has a tendency to increase on most benchmark data neglecting the involvement of AM for a moment. This rise, however, is very flat and never reaches the full classification competency as before the drifts at least using the data at hand. Moreover, a rather high number of anomalous FVs is observable in most cases (see *ADR* increase without AM resolution in Figure 10.21). Both effects are expected. On the one hand, emerging new structures in the data are expressed by a growing *ADR*, which has a positive impact to the classification performance on the other hand as the anomalous class anticipates cases that are tough to classify by the HFIDS. Despite this effectiveness to identify new shapes in the data, such situations are undesirable in practical intrusion detection particularly when *ADR* reaches an unacceptable high proportion. These uncertainties can and should be addressed by AM. With this component in place, growing *ADR*s can be brought down largely and classification performances increase eventually such that we obtain a combined $mF1_b$ of 87.33% right after the drifts. In this respect, further points are worth to mention. Even though we involve AM, a direct performance rise is not guaranteed necessarily after each resolution. This behavior is due to the reactive nature of our system that can adjust to data it has seen but subsequent data chunks might or still might not be reflected by underlying models. As the HFIDS experiences more data from a fixed distribution, these fluctuations in the performance decline. Mostly, this effect is discernible at the end of each benchmark data set because distributions become stable again. Additionally, no major differences are observable either using feature combination F00 or F0W in terms of peak performances, which can be related to the sizing of involved benchmark sets. On larger data sets, though, considerable differences between those FFC can be expected (see Section 10.3.4). In turn, a clear contrast is notable applying F00 and F0W on DS-II-to-I-100k and DS-II-to-I-10k. Despite various resolution steps, the system has difficulties to regain the same or similar classification potentials after the drifts employing F00 whereas the utilization of F0W does not show this tenacity. We attribute this finding to both the superior expressiveness of combining flow features with traffic statistics over pure flow features and the level of difficulty comprised by these data sets. While the former point could already be unveiled in Chapter 7, the latter aspect is more comprehensible when reviewed from an incremental learning perspective. A model that already explains well DS-I can be adapted to DS-II with certain effort recalling that DS-I is more competitive than DS-II in the sense that more attack diversity is comprised (see Figure 10.12). Yet twisting the situation, the adaption process becomes clearly more challenging because a higher degree of novel attacks is contained that are obviously harder to detect. This effect can also be explained by resurveying F00's performance on the simpler data sets DS-I-to-II-100k and DS-I-to-II-10k where percentages after the drifts converge rather quickly to numbers obtained before the drifts. Combined, this outcome demonstrates that in six out of eight benchmark tests, the HFIDS is capable to recover after different drift scenarios and even in the other two experiments a tendency to regain old performances is given all with moderate supervision. In this context, *ADR*s can also be kept low to acceptable levels with values of not more than 3% on average. $mF1$ is clearly weaker compared to its binary counterpart $mF1_b$ but can be improved slightly when using combination F0W

Data set	F00			F0W		
	$mF1$	$mF1_b$	ADR	$mF1$	$mF1_b$	ADR
DS-I-to-DS-II-100k	68.0354	91.0056	0.0502	69.9872	88.8422	0.0001
DS-I-to-DS-II-10k	66.9528	92.4150	0.1025	69.7658	90.7083	0.8327
DS-II-to-DS-I-100k	62.4212	83.1706	1.4023	65.9347	88.0308	2.9212
DS-II-to-DS-I-10k	61.2251	82.7181	0.7315	73.8731	90.5329	0.9525

TABLE 10.3: Average classification result of the HFIDS on all four synthetic drift data sets per FFC with AM resolutions right after the drifts; all measurements are expressed in percent

FFC	Data set	ABR	size	len	mult	cov	pur
F00	DS-1-to-DS-2-100k	28.1438	643.2212	14.0151	0.0013	34.5221	73.9823
	DS-1-to-DS-2-10k	31.8708	698.2301	14.4511	0.0007	34.1416	67.8319
	DS-2-to-DS-1-100k	63.1124	1910.5487	15.6926	0.0015	32.3363	39.8053
	DS-2-to-DS-1-10k	56.4982	1695.9823	15.6542	0.0023	32.6195	41.8761
F0W	DS-1-to-DS-2-100k	18.5617	5319.2124	18.9841	0.1622	68.5900	61.6578
	DS-1-to-DS-2-10k	27.4958	7494.6106	19.8272	0.1975	64.8850	54.8673
	DS-2-to-DS-1-100k	29.6464	8909.7080	19.8384	0.2519	64.6416	53.3761
	DS-2-to-DS-1-10k	27.7106	7047.1327	19.4203	0.2273	64.0442	65.7434

TABLE 10.4: Pattern-related aspects on all four synthetics drift data sets with respect to the average ABR of PMC, PDB size (size), rule length in PDB (len), multi-match rate at PMC (mult), coverage of PDB (cov) and rule purity (pur) per FFC; ABR, mult, cov and pur are expressed in percent

over F00. A supplemental summary of those obtained classification performances is outlined in Table 10.3. In terms of pattern-related aspects, a relative high ABR is observable for those challenging benchmarks DS-2-to-DS-1-100k and DS-2-to-DS-1-10k using F00 where PMC's average abstaining affects 59.80% of the cases. Considering the other six experiments, a much lower abstaining behavior is notable. On average, the system classifies 72.76% of the data with a transparent decision. Additionally, a remarkable result is obtained in terms of multi-matching. For both FFCs, the multi-match rate is below 1.00% such that the chance of ambiguity in those transparent decisions is very low. These and further discovery-oriented measurements obtained during the synthetic drifts scenarios are pictured in Table 10.4.

10.3.4 Assessment on Enterprise Traces

Previously, the HFIDS was assessed under different synthetic drift scenarios where we managed to adapt the system to new circumstances with suitable operating points in place and the support of AM. In this section, we basically employ the same setup but extend the analysis by applying our solution on a large enterprise network capture that is salted with a broad spectrum of different attacks, i.e. DS-III. As it is uncertain whether this very representative benchmark data set contains drifts, we strive for an experimental setting that leaves out supervision in the live phase in analogy to EVL. This way, a pessimistic view is obtained to the capabilities of the HFIDS. It is pessimistic in the sense that by including AM as instrument to resolve anomalous clusters, it can be expected that the overall performance is at least not weakened, which was already demonstrated

FFC	Approach	$mF1$	$mF1_b$	TPR	FPR	Rank
F00	aNB	44.6848	69.3819	20.4142	0.0432	4.00
	DWM	47.2328	73.0216	21.2153	**0.0088**	2.75
	HAT	57.2418	76.6655	33.5375	0.0155	2.25
	HFIDS	**69.6339**	**82.7038**	70.2320	0.1252	**2.00**
	VFDR	33.5268	62.9884	**90.1369**	21.8351	4.00
F0W	aNB	45.0296	71.1373	20.5400	0.0276	4.75
	DWM	**89.8414**	**95.9298**	86.0661	**0.0025**	**1.25**
	HAT	84.1998	94.6511	85.4097	0.0127	2.50
	HFIDS	84.7845	93.8541	**86.7067**	0.0238	2.25
	VFDR	55.5154	71.5835	64.4617	0.5474	4.25

TABLE 10.5: Average classification result of the HFIDS on DS-III per FFC including the outcome of state-of-the-art incremental learners as reference; $mF1$, $mF1_b$, TPR and FPR are expressed in percent; bold numbers indicate overall winner per performance measure and FFC when incorporating a mild comparison

in prior assessments. We also involve the conventional performance metrics TPR and FPR to get a better understanding of what actually improves or deteriorates when DS-III is applied on different FFCs. The second part of this assessment deals with transparency. We state discovery-oriented aspects and highlight some representative attack patterns that are exposed. Lastly, this section not only provides an isolated assessment of our solution but also a comparison with other state-of-the-art algorithms.

We begin with the predictive quality obtained by applying our HFIDS on DS-III with fixed operating points after the initial training phase (i.e. the first 30000 FVs) using F00 as FFC. In this EVL setting and employing pure flow features, results reveal that our system has a moderate overall multiclass prediction performance outlined by $mF1$, which is roughly 70% on average over the course of DS-III. Its binary classification is better though using this FFC. This is reflected by a $mF1_b$ that reaches approximately 83%. Remarkable in this regard is the low false alarm rate which is below 1% on average. This is a very positive outcome when reviewed from a credibility standpoint. Yet, the downside of utilizing pure flow features is the poor hit rate. It is around 70%, which can be considered devastating at first sight because roughly one out of three attacks, in fact, is missed. Turning over to F0W where also statistics of adjacent flows are taken into account, the outcome of the HFIDS is much better in general. Its multiclass capabilities increase by around 15% and also the binary performance improves in a similar fashion such that we obtain a raised score of 11% on average. This clear enhancement has its roots in both the TP and the FP. While roughly six out of seven malicious FVs are correctly predicted using F0W, there is a notable 5.25-fold decrease in raised false alarms. This means that only 240 out of a million benign FVs are falsely classified. These reported results are outlined as part of Table 10.5 whereas the continuous performance over the course of DS-III is showcased in Figure 10.22 employing $mF1_b$ as measure. The detection of anomalous FVs is also in the order of counted FP considering F0W where appropriately 0.03% of the data is classified as such. On F00, the percentage of identified anomalies is clearly higher on average ($ADR = 1.55\%$). This rate starts to grow rapidly after having processed two million FVs. From there, it oscillates between 1.50% and 2.00% until the end of the benchmark test, which is not surprising given that AM is excluded intentionally in these experiments. Concrete averages of ADR on

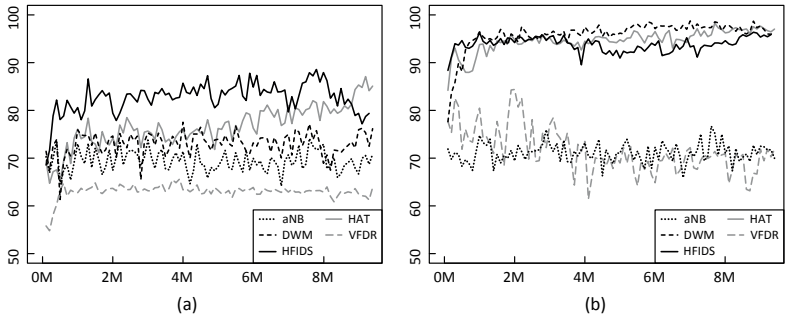

FIGURE 10.22: Classification performance of the HFIDS using $mF1_b$ in percent over the course of DS-III per FFC including the outcome of state-of-the-art incremental learners as reference: (a) F00 and (b) F0W

FFC	*ADR*	*ABR*	mult	size	len	cov	pur
F00	1.5548	7.5620	0.0112	7190.9680	9.1922	58.8979	70.0143
F0W	0.0259	4.1107	0.0074	6961.8280	11.6270	85.4801	84.4452

TABLE 10.6: Detected anomalies and pattern-related aspects with respect to *ABR* of PMC, multi-match rate at PMC (mult), PDB size (size), pattern length (len), pattern coverage (cov) and pattern purity (pur) on average over DS-III per FFC; *ADR*, *ABR*, mult, cov and pur are expressed in percent

DS-III per FFC are outlined in Table 10.6. In the context of predictive performances, it is also interesting to analyze which attack classes can be detected surely and those which are rather difficult to unveil. To provide insights into this question, we utilize the confusion matrices in Figure 10.23 illustrated as heatmap that relate the GT of DS-III to the complete classification result. Using combination F00, the graphics attest solid detection results for brute-force and spoofing attacks where over 95% of these classes are correctly predicted. The detection of flooding attacks such as DoS or DDoS is at the borderline with roughly 79% true classifications. All other attacks are hit within a range of 17% to 25% only. The exception are malware/exploit instances where as little as 2% of their predictions are actually correct. On the contrary, specific attack detections per class can be increased significantly utilizing F0W. Assessing brute-force and spoofing attacks, their recognition is still reasonable with similar scores than reached by F00 but a small portion of brute-force instances are mixed up with other attack types. The detection of floodings can be increased by roughly 16% such that this attack type is no longer at the borderline and, thus, the identification of DoS and DDoS attacks can be considered certain. Exposing other malicious activities such as C2 communication, vulnerability scans or spam can be enhanced by around 44% on average such that their chance of being detected lies in the range of 65% to 70% given that such attacks are actually present. Hence, they get to the borderline as their coverage becomes moderate. The last two classes malware/exploit and web attack including ransomware, exploits, SQLIs and XSS are still difficult to be identified despite improved hits to roughly 40% on average. Web attacks are worth to stress at this point because a remarkable 44% of prediction attempts on that class, in fact, are determined to be botnet instances.

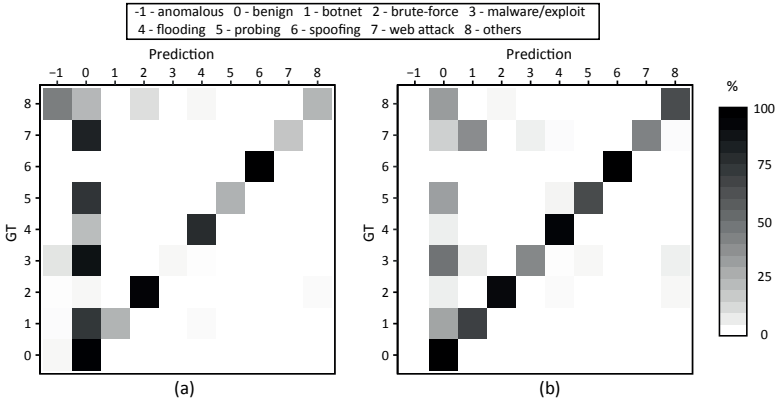

FIGURE 10.23: Detection capabilities of the HFIDS per attack type and FFC on DS-III illustrated as heatmap where each row sums up to 100%: (a) F00 and (b) F0W

On the one hand, this indicates that our system has difficulties to separate well both malicious classes. Putting these observations into perspective, on the other hand, reveals that effectively 86% of web attacks can be detected with certainty when we turn from multiclass classification to a binary problem. Thus, predicting web attacks becomes also reliable employing flow features together with traffic statistics.

Apart from this predictive evaluation, quality aspects are important as well especially when it comes to transparent decision-making. Therefore, we want to present pattern-related characteristics and describe how they evolve during the processing of DS-III. Beginning with the abstaining behavior as indicator to which extent the HFIDS is incapable to provide a proper explanation for a given decision, results are two-fold from start. While abstaining on F00 is fairly high during the processing of the first two million FVs in DS-III catering for an average of roughly 11%, it converges rapidly to an average of 6% afterwards right to the end of the benchmark test. On F0W, the *ABR* is 3.66 times lower initially and slowly raising to similar numbers as for F00 but never surpassing them. This acknowledges that *ABR* becomes smaller than or equal to 6% after some initialization period for both FFCs. Directly related to situations where PMC is not abstaining is the multi-match rate. It is below 0.02% on average either utilizing F00 or F0W, which can be considered small. That given, let us move to the sizing of PDB and the length of patterns. During the processing of DS-III, PDB grows slowly from 4000 to 8000 patterns when using F00. In terms of F0W, a growth is noticeable as well starting roughly from 2000 to 10000 patterns. Yet, its intensity is much steeper when processing those FVs in the range of two and six million. This strong increase is a typical indicator for a knowledge gap and possible retirement activities resulting in the creation of new patterns. On the other end, the average pattern length drops significantly after processing the first million FVs given our bottom-up induction approach and it becomes relatively stable for the next two million FVs on both FFCs. While a constant average length below ten is maintainable until the end of the benchmark for F00, an increase is observable after processing three million FVs using F0W. It can be related to the

FFC	ID	Pattern condition and consequent	$cov(p_j)$	$1-\delta(p_j)$
F00	p_1	$\big((4900 \leq a_1 < 5100) \wedge (a_3 = \text{'HTTP'}) \wedge (a_{26} = \text{'SF'}) \wedge (1.70 \leq a_{28} < 1.71)\big) \Rightarrow (a_{57} = \text{'web attack'})$	28	92.86%
	p_2	$\big((a_3 = \text{'HTTP'}) \wedge (422 \leq a_6 < 573) \wedge (4911 \leq a_7 < 5695) \wedge (a_{12} = 0) \wedge (a_{13} = 1) \wedge (a_{14} = 1) \wedge (a_{15} = 1) \wedge (a_{26} = \text{'SF'})\big) \Rightarrow (a_{57} = \text{'brute-force'})$	278	100.00%
	p_3	$\big((a_3 = \text{'HTTP'}) \wedge (5696 \leq a_9 < 5729) \wedge (343 \leq a_{21} < 367) \wedge (a_{26} = \text{'SF'})\big) \Rightarrow (a_{57} = \text{'botnet'})$	30	100.00%
F0W	p_4	$\big((a_3 = \text{'NoSC'}) \wedge (a_{26} = \text{'REJ'}) \wedge (95.00 \leq a_{44} \leq 100.00) \wedge (184 \leq a_{45} \leq 200)\big) \Rightarrow (a_{57} = \text{'probing'})$	602	100.00%
	p_5	$\big((a_4 = 1) \wedge (a_{26} = \text{'S1'}) \wedge (995 \leq a_{49} \leq 1000) \wedge (99.00 \leq a_{50} \leq 100.00)\big) \Rightarrow (a_{57} = \text{'flooding'})$	2205	99.95%
	p_6	$\big((a_3 = \text{'ARP'}) \wedge (a_4 = 4) \wedge (5 \leq a_{49} < 9)\big) \Rightarrow (a_{57} = \text{'spoofing'})$	121	98.35%

TABLE 10.7: Selected patterns exposed during the processing of DS-III with coverage and purity evaluated at the very end

mentioned growth of PDB such that we obtain an average pattern length of 14 for the last five million FVs. To evaluate the pattern coverage and purity, we employ a reference sliding window with a constant size of the last 5000 FVs processed. As such, we reach very stable coverages over DS-III, which is roughly 59% using F00 and 85% utilizing F0W. The purity in turn fluctuates more. Particularly on F0W, a decrease from roughly 95% to 80% is noticeable handling the FV range between two and four million, which supports aging effects and the inability to reflect the most recent situation. Anyhow, it should be stated that due to the creation of new patterns, the decline is absorbed such that the average purity becomes constant again. An aggregated view of these discovery-oriented characteristics is provided in Table 10.6. In light of these quality aspects, we also want to showcase three concrete decision rules per FFC exposing malicious activities with coverage and purity given through Table 10.7. They are generated during the processing of DS-III and are among the most general patterns found in the data. Even though our pattern engine only supports a very basic form of propositional rules using the equality comparator (see Definition 9.1), more expressive literals are found in that table employing arithmetic operations such as "$<$" or "$\leq$" in addition. This is due to the underlying discretization process permitting to apply the equality operator on data ranges and so no extended functionality is added. Moreover, we have to get a naming convention for those attributes comprised in each of these patterns. Therefore, we make use of those feature IDs introduced in Chapter 7 and write a_i to correspond to the i-th discretized attribute in Table 7.3 where $1 \leq i \leq 57$. Starting with F00, the first pattern we want to outline is p_1 targeting SQLIs and XSS. Basically, it states that HTTP flows with handshake and tear-down lasting for a duration of roughly five seconds and having an average interarrival time of 1.70 seconds in reverse direction are suggested to be web attacks. This pattern is effective over a long period in DS-III. Yet, at the end of the benchmark two counter example are observed such that it accounts for a purity of 92.86% ultimately. An interesting point for web attacks is the fairly consistent average interarrival time of packets in both directions (see a_{27} and a_{28}). It is the most homogeneous among all eight attack classes in DS-III, which is why PBC considers these two features to describe web attacks. We attribute this intrinsic characteristics to underlying attack tools employed. By using other more sophisticated programs, it

becomes clear that these attacks can be obfuscated easily such as establishing random delays between packets or other sorts of evasion tactics requiring better engineered features. Such features would be a_{37} or a_{38}, for instance. Pattern p_2 aims at HTTP brute-force attempts. It triggers if an ordinary HTTP flow is seen with a total forward octet volume between 422 and 573 and 4911 to 5695 octets in reverse direction from which only one packet essentially contain payload in either direction. This behavior is exceptional because normal HTTP flows transporting data in the specified ranges usually distribute the load more evenly among the packets especially in backward direction. This concrete attack pattern represented by p_2 is also contained in other rules as a result of the discretization (see cut-points for a_6 and a_7), which manifests that described characteristics hold for different octet ranges as well. Further variants of p_2 involve a_{27} instead of a_{12} to a_{15}, which is also a valuable pattern due to the opportunistic nature of the HTTP brute-force attack trying to issue and wind up login attempts as fast as possible. As such, the replacing literal $a_{27} = 0.00$ is valid in some situations having the advantage of a lower rule complexity. However, such patterns are susceptible because the overall network load can influence interarrival times considerably rendering them ineffective unless no network system saturation is reached. This observation also applies to p_1 consequently. The last pattern we want to underline for combination F00 is p_3 and addresses C2 botnet communication via HTTP of an arranged Citadel infrastructure. It states that ordinary HTTP flows carrying a payload between 5696 and 5729 bytes in reverse direction where the first packet in that direction has a length in the range of 343 to 367 bytes are considered to be botnet instances. This described footprint is precise at least with respect to our data source because no other HTTP flow transferring similar amounts of data occupies the specified payload length in the first packet resulting in a coverage of 30 FVs with a purity of 100.00%. Recalling that one of the major flaws of F00 is not only related to the coarseness of flow data but also to missing context regarding other FVs that arrive concurrently (see argumentation stated in Section 7.3.2), we turn to patterns generated based on F0W. The benefit of this FFC becomes especially apparent when reviewing probing attempts for a moment. Considering the state of a flow alone can provide a clue about an ongoing port scan (see a_{26} = 'REJ') but rating it as harmful is overoptimistic because such flows can also be a result of legitimate actions. To substantiate suspicion, the incorporation of traffic statistics can be supportive where we can sense whether a manifold of those actions take place or not and, in fact, a large portion of probings can be unmasked with this technique. A representative pattern for that matter is p_4. It matches FVs with no SC that are rejected by the responder where 95.00% to 100.00% of concurrent flows to the same destination have a similar state and the number of recent FVs seen with no SC in place lies between 184 and 200 in our environment. p_4 covers 602 FVs out of which all are classified correctly. Remarkable to this exposed pattern is its generality as it also fires successfully on a large amount of malware infiltrations and DoS attacks in DS-II without raising any false alarm. The next pattern p_5 targets SYN floods. Characteristic for this type of flooding is to bind resources at the victim through half-open connections, which is signaled by the fact that only one packet is seen in forward direction combined with the 'S1' state. Besides this footage included in p_5, it also inspects whether the responder is frequently penetrated in the last ten seconds out of which a high percentage of SC corresponds with the actual flow finally concluding flooding actions. The coverage of this pattern is fairly high where only one FV is negatively covered resulting in a purity of 99.95%. Yet, the

negative match is not a severe issue because that single flow instance is part of another attack class. Lastly, we want to point out a simple pattern for spoofing attacks, i.e. p_6. It rates ARP FVs comprising four packets in forward direction as spoofing attempt when a moderate number between five to eight FVs are seen in the last ten seconds to the same destination. p_6 covers 119 FVs correctly whereas two instances are actually vulnerability scans catering for a purity of 98.35%. Similar effective patterns are also observable targeting ARP spoofings where the ranges of a_{49} are marginally higher, which is also an effect of the underlying discretization. Hence, a rule engine capable to process continuous data would be clearly beneficial as more expressive and more compact rules could be produced.

Given the discussed results, we devote the remainder of this section to a comparison by contrasting the HFIDS with other existing approaches using DS-III. Even though it is extremely difficult to find competing systems that are on par with our solution (see Section 10.3.6), we managed to compile a lineup of four state-of-the-art ML algorithms known from the stream mining community that partly share aspects of our holistic system. This lineup includes an "adaptive NB learner" (aNB) that uses an explicit drift detector (see [368]) instructing to retrain the underlying NB model once triggered. Additionally, DWM is applied with VFDT as base learner that is very close to our ensemble. We also incorporate the known rule learner VFDR [98] and HAT [483], which is an enhanced version of VFDT with a dynamic sliding window (see [435]) adapting to potential drifts implicitly. All implementations are taken from MOA and default parameter values are utilized unless better values were documented. With this competing lineup in place, it should be stated that due to the different concepts of those learners and our system, the confrontation is generally biased. Our solution operates under EVL and the other four algorithms are designed for prequential setups (see Section 9.4). This means that the HFIDS has the opportunity to sense only the first 30000 FVs with underlying GT for training while class labels remain basically hidden during the live phase. The other inducers in turn have full access to the GT of each and every FV in the live phase right after it is classified. Based on these differences, results should be interpreted with caution after all. Hence, we refer this examination to be a rather mild comparison. With these prerequisites in mind, let us turn to the assessment where we report about average results split by FFC. Using combination F00, VFDR stands out in terms of attack detection capabilities. Its TPR is 90.14% on average, which is by far the best among all five learning algorithms. In this category, our HFIDS is the second best where 70.23% of the attack instances can be uncovered with certainty. All other systems are clearly behind with a $TPR < 35.00\%$. These solid detections of VFDR, however, come at a price. It also produces a very high amount of false alarms ($FPR = 21.83\%$) rendering this solution ineffective in practice. In contrast, all other approaches raise significantly fewer undesirable alerts of that type with a $FPR \ll 1.00\%$ on average. In terms of multiclass performance, our HFIDS reaches superior results over its contenders. It is ahead by 12.39% to the second place. Additionally, its balanced binary prediction capability is the most sound. A roughly 6.04% higher $mF1_b$ is achieved than by the second best solution. As such, the HFIDS is the best rank-wise followed by HAT and DWM. Due to the weak $F1$-scores and the high number of false alarms, VFDR is only in fourth position that is shared with aNB. Considering the utilization of F0W on DS-III, results are closer. DWM caters for the best $F1$-scores with $mF1 = 89.84\%$ and $mF1_b = 95.93\%$ on

average. The HFIDS is behind by a difference of 5.06% in terms of multiclass prediction abilities (second place) and 2.08% when considering binary classifications (third place). Yet, its attack detections are the best of all learners. With respect to false alarms, our system is in third position. Combined, our system comes in second rank-wise right after DWM. HAT, VFDR and aNB trail to our HFIDS in consecutive order. These reported results are outlined in Table 10.5. Besides the comparison of pure numbers, let us also look at the performance developments over the course of DS-III, which are depicted in Figure 10.22. In Figure 10.22 (a), it is observable that the $mF1_b$ of the HFIDS is relatively stable whereas the trend of HAT is increasing such that it outpaces our system at the end of the benchmark test eventually. This result is, somewhat, remarkable because HAT requires more than eight million FVs with underlying GT to achieve this goal, which is essentially the amount of data produced on an entire working day. On the contrary, our system requires only 30000 FV with GT in the warm-up phase. On the one hand, it is arguable that the reason why the performance of the HFIDS can be maintained over the course of DS-III is due to the notion of ADR that preserves our approach from making wrong predictions on difficult FV instances. On the other hand, we are quite confident that with minor supervision ADR can be brought down as demonstrated in Section 10.3.3 and the classification performance can be boosted or at least maintained. From this perspective, our solution can be considered clearly better using pure flow features. When incorporating traffic statistics given through Figure 10.22 (b), it is noticeable that the advantage of ADR vanishes. In this setting, a very low ADR is obtained ($< 0.03\%$) and so DWM, HAT and HFIDS are getting closer performance-wise. Unfortunately, aNB and VFDR are still far off. These results acknowledge two important finding: (i) our system can compete with other approaches that rely on complete GT availability with (ii) the amenity that transparency is supplied in at least 92% of the cases on average ($ABR \leq 8\%$). This is in contrast to DWM and HAT whose predictions are more complex to analyze and interpret from an operative standpoint. Lastly, we want to give a brief grasp when porting aNB, DWM, HAT and VFDR closer to our EVL setup where the four competitors are only trained on the first 30000 FVs of DS-III, which provides insights to the benefits of internal adaptions during the live phase. In this respect, the HFIDS outruns them substantially in terms of $F1$-scores for both FFCs. On F00, multiclass predictions are better by 23.49% and the gap considering binary classifications is 10.12% to the best of its condensers, i.e. DMW in this comparison. Employing F0W, the second best approach is HAT. It obtains a $mF1$ that is 23.06% lower than the output of our system and a 5.86% weaker performance on $mF1_b$. As such, we can also conclude that our continuous attempt to adapt to the data pays off and has a positive impact to prediction quality in general and for multiclass tasks and correct attack detections in particular. The latter point is additionally manifested comparing TPRs. Our hit rate is certainly better on both FFCs. It is 47.49% higher than the one of HAT utilizing F00 and 26.76% higher than the outcome of DWM when using F0W (both in second place for the TPR and respective FFC).

10.3.5 Scalability Potentials

Earlier system evaluation aspects were mainly concerned with the assessment of predictive and descriptive capabilities. Yet, another crucial factor, neglected so far, is the

HWC	Kafka	Flink	RAC	vCPUs	RAM
H1	1	1	2	28	84
H2	2	2	2	40	120
H3	4	4	3	72	216
H4	6	6	3	96	288
H5	8	8	4	128	384
H6	10	10	4	152	456

TABLE 10.8: Defined HWCs of the cluster lineup to assess data throughputs at scale with number of nodes per VM type, cumulated amount of available vCPUs and RAM (in Gbyte)

continuous growth of traffic volumes and velocity caused by the deployment of faster network hardware or by expanding the existing network infrastructure with new segments. These circumstances carry practical implications for any modern NIDSs as more data needs to be monitored and analyzed in a single time frame. This in turn requires appropriate scaling abilities in order to cope with rising throughputs. Therefore, an empirical study is provided in this section to get insights on this particular subject with regards to the most important components of the HFIDS considering all four FFCs. We start this analysis by illuminating experimental preliminaries including hardware profiles, workload, measurement points, host data generation as well as several other details. On these grounds, we present scaling results by metering attainable throughput rates in terms of FVs per second (FVs/s), obtained speedups per FFC (as they involve different tasks in MIP) and emphasize the hardware utilization of the most promising FFC. Finally, we reuse throughputs gathered during our experiments and extrapolate them to hardware configurations (HWCs) that exceed our test environment.

In order to state the scaling potentials of the HFIDS inside the given virtualization, several practical prerequisites have to be taken into account beforehand. These include the identification of suitable hardware alignments along the defined VM profiles of Table 10.1. With respect to the provisioned quota policies inside that virtual testbed, a multitude of test trails supported us to find such a setting. It contains six successive HWCs fostering our objectives to systematically assess different variants of the cluster lineup consisting of the three subsystems Kafka, Flink and RAC whereas the settings for producer, ZooKeeper, ADC (offline) and AM remain constant. These profiles are outlined in Table 10.8 beginning from H1 as starting point up to H6 forming the lineup with the most powerful hardware resources that could be built with our available hardware contingent. Despite these pretest results, it should be stated that we also experienced unexpected processor peaks for individual Flink nodes raised by other system processes. Such situations can cause undesired overloads and degrade cluster throughput rates dramatically if all eight vCPUs of a Flink VM are assumed to perform the actual stream processing. For that reason, we restrict the number of vCPUs for each node to only six, while the remaining two processing units are reserved for TT cache and potential system overloads just mentioned. Another important point is concerned with the actual workload serving the scalability analysis. In order to provide realistic conditions, we reuse DS-III constituting a convenient fit in terms of data volume. Yet, we are not simply replaying the data with original timing, which would result in a very similar flow distribution as given in Figure 10.14. Instead, all comprised flows are pushed to Kafka in

a sequential order as fast as possible using the producer VM that operates as simulated IPFIX exporter in our setup. As a consequent, a consistently high flow frequency can be acquired in comparison to the original profile showcasing a sustainable stress test for our IDS. To regulate the data volume between producer and Kafka, a sleep mechanism is employed to either slow down the consecutive flow transmission or to immediately pass them through. Putting together the given HWCs and the considered workload leads to a further challenge that is related to the establishment of measurement points tapping throughput rates adequately. Even though the combination of all three subsystems Kafka, Flink and RAC contribute to the efficient processing of incoming flows, they are running independently. Thus, they split the global system functionality logically in a distributed manner involving data exchange activities such as grouping or rebalance actions that impede the utilization of conventional in-line measurement methods. Despite these circumstances, two distinguished observation points can be identified on a global level with no or very low latency effects during monitoring, i.e. the producer and RAC. Given that the producer specifies and controls the amount of raw flows entering the IDS, RAC functions as final data sink. As a result, we are able to measure the throughput at the producer and can contrast it with throughputs obtained at RAC using the arrival timestamps of persisted FVs at TDB and the required processing time at PBC. In that sense, all of these rates have to correspond necessarily in order to yield near real-time processing abilities[12]. Strong deviations of these measures demonstrate that the system is highly strained and not capable to carry the ingress load of the producer appropriately, which suggests to append more hardware to the cluster lineup ultimately. In addition, we have to recall that DS-III relies solely on network traffic without any host information. This poses an issue because a mechanism has to be devised to test the efficiency of the correlation when using FH0 or FHW respectively. Therefore, we enhanced the producer VM with an additional host data generator that produces such data synthetically in a reasonable quantitative manner. It selects every fifth flow to be both an IPFIX flow and a related host event. Thus, one message is sent for each of the first four flows while two independent messages are created for the fifth that are finally pushed to their designated Kafka topic. For the purpose of simulating occasional delays between both data sources, the messages cannot be sent simultaneously as that would result in an ideal processing situation otherwise. Instead, emerging host information are transferred with a random delay of 20 up to 80 seconds making the correlation task more realistic and competitive at the same time. At this point, we should note that from a quantitative perspective this approach comes close to conditions we experienced in practice but certainly it is too simple to provide quality insights. On that account, host features cannot be employed for any predictive model in the remainder of this section. Further elaborations in this direction disclose that the scalability assessment is not relying on any initial training phase to build up models because fewer data would be at hand for the actual objectives. Based on that, we skip training and inject predefined models as starting point. In the same context, PDB and its counterpart residing at Flink (see local PDB) is left empty intentionally such that the pattern building commences from scratch. All other parameters of the HFIDS are inherited from Section 10.3.2 where missing operating points for FH0 and FHW are taken from their respective base FFC, i.e. F00 and F0W. A final remark points towards the basic setup of Flink and Kafka.

[12] Exceptional to these capabilities are the correlation attempts (see FH0 and FHW) where a FVs resides up to 60 seconds in MIP (see Section 10.2.3).

Generally, we employ both systems in default mode whereas Flink uses slot sharing and the coordinating master resides on the first Flink node (i.e. Flink-1). A closer analysis with respect to fault tolerance that can be enabled explicitly via "replication factors" [484] for Kafka topics or via "checkpoints" [485] at Flink is neglected. This is due to the fact that additional resources would be required to enable these mechanisms, which is critical to our already confined testbed hardware contingent.

Using these considerations, we outline the average scaling results that are obtained based on five repeating test runs. They reveal a nearly linear throughput behavior for all four FFCs as hardware increases. Yet, different intensities can be identified starting from the initial configuration H1 all the way to H6. In this respect, the most promising performance is reached by F00 with an average throughput of 4688.01 FVs/s at H1 up to an average rate of 20195.21 FVs/s contemplating H6. Putting these numbers into perspective with the other three FFCs demonstrates that F00 is actually better by at least 1.20 times than FH0 on H1, which is expanded to 1.79 times considering H6. This means that F00 is capable to process 8894.87 FVs more that FH0 in a single time frame. In comparison to F0W and FHW, F00 creates an even greater performance gap with a throughput that is at least two times higher employing H6. These results are not surprising given the straightforward processing pipeline of F00 within Flink. It only relies on a single IPFIX flow to create, classify and persist a new FV. Hence, no dependencies among the data need to be considered implying significant fewer operation and stream repartitioning efforts. This is in contrast to the other FFCs entailing additional costs for flow and host data correlation[13] (see FH0 and FHW) or the computation of traffic features (see F0W and FHW). In this context, it is interesting to weight the overhead caused by the correlation in comparison to the calculation of traffic features. For this purpose, we can compare the outcome of FH0 and F0W. Pairwise inspecting them indicates that F0W has a tendency to require slightly more resources in order to acquire similar throughputs as FH0. However, this difference is marginal in practice because measured rates on both FFCs vary only between 1123.10 FV/s and 1441.86 FV/s in the range from H3 to H6. This way, we can assess their computational overhead to be fairly similar but should keep in mind that with substantial more resources added to the cluster lineup, gaps between FH0 and F0W are expected to spread constantly. Turning to achievable speedups in our test environment underpins the reported dominance of F00 over the other FFCs in addition. With a maximum average speedup of 4.31, F00 maintains the best outcome followed by FH0 that has a 2.90 times higher throughput compared to its initial results at H1. The performance increase of F0W is slightly smaller than FH0 with a speedup of 2.62. The poorest result during our experiments is obtained by FHW with a speed enhancement of only 2.17 contrasting its rates at H6 and H1. A complete depiction of this assessment is given in Table 10.9 and Figure 10.24 showing conducted throughputs for all FFCs alongside with corresponding speedups per HWC.

In addition to these findings, we want to highlight the hardware utilization of our proposed HFIDS in more detail exemplified by F00 at H6 as this FFC demonstrates the best scaling potentials and the highest throughput rates during our experiments exemplified by Kafka, Flink and RAC while neglecting ADC (offline) and AM as they only

[13] For FH0 and FHW, we obtained a successful correlation rate of 84.20% on average during our tests using the host data generator introduced in the previous paragraph.

HWC	F00	F0W	FH0	FHW
H1	4688.01	3798.13	3900.09	3602.02
H2	6652.80	4343.01	5050.33	4219.01
H3	10627.21	5439.20	6824.67	5666.27
H4	13992.14	7428.81	8870.67	6150.09
H5	16332.52	8976.83	10099.93	7400.17
H6	20195.21	9947.22	11300.34	7802.06

TABLE 10.9: Average throughput rates in FVs/s obtained per HWC on all four FFCs applying DS-III

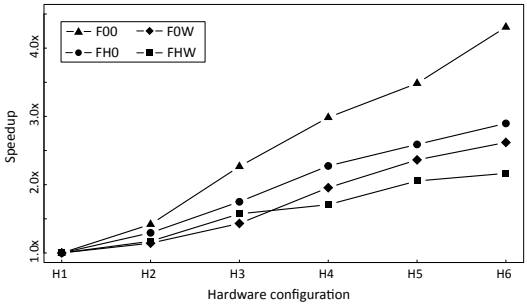

FIGURE 10.24: Average speedup of the HFIDS on all four FFCs applying DS-III

can scale-up. Hence, we start with the workload of roughly 20000 flows per second generated by the producer VM arriving constantly to the Kafka cluster. In this context, it is observable that the pushed content is well balanced among all ten available brokers of the messaging queue, which is due to the partitioned flow topic sharded across the cluster. This discloses that approximately 2.12 Mbytes per second are received at the network interface of each Kafka node catering for an overall cluster throughput of 21.20 Mbytes per second. Despite of this ingress traffic volume, the processor utilization of Kafka is fairly low such that only 15.19% of all 40 vCPUs are in use on average. Yet, the distribution of that load is not as consistent as given by the reported input traffic of the cluster because the first Kafka node (i.e. Kafka-1) stands out with a higher mean utilization and local peaks nearby 35% that arise occasionally. All other nodes are very close to the given cluster mean and, therefore, constituting a balanced usage. We attribute this observation to coordination effort taken by Kafka-1 to keep the cluster in sync with the latest topic offsets. Concerning the main memory consumption, Kafka initially requires around 10.50 Gbytes per node on average and demands constantly grow over the course of processing but never surpass 12.10 Gbytes, which is below the maximum available resources of 16 Gbytes per node. Hence, its memory usage can be rated rather low. Based on these two last findings, a reasonable adjustment to Kafka's VM hardware profile is worth to consider such that the cluster would still be able to process the same amount of data but with a reduced number of vCPUs and memory resources respectively. Further details about the reported hardware utilization at Kafka are illustrated in Figure 10.25 (a). In order to expose the consuming part of those buffered IPFIX flows at Kafka, we turn to the Flink cluster. It consists of ten nodes and reserves six out of eight available vCPUs per node exclusively to realize MIP recalling

that the two remaining vCPUs are intended to absorb unexpected peaks raised by system processes. However, these overloads only play a marginal role as they are absorbed by the presented averages such that the maximum processor utilization does not exceed 67.74%, which is very close to the ideal utilization of 75%. In terms of memory usage, Flink initially requires 6.66 Gbytes per node and demands increase up to 11.38 Gbytes on average in order to process the entire workload. Thus, roughly 8.50 Gbytes per node remain untouched. In this respect, a reassessment of Flink's hardware profile should be considered to some extent in a similar way as for Kafka VMs. The analysis of its network interface bares that ingress traffic at Flink is partly twice as high compared to the egress traffic of Kafka and, in fact, five Flink nodes receive a proportionally greater amount of data. This outcome can be justified along two aspects. On the one hand, ingress traffic is produced by the incremental update mechanism implemented by RAC and TT cache to deploy, modify or even remove patterns at local PDBs consequently (see Section 10.2.4), which becomes observable by the traffic peaks of Flink's input. The stronger impact with respect to the ingress traffic can be associated with the stream splitting activity of the Flink program on the other hand (see Section 10.2.6). As such, a degradation of the parallelism is entailed because two sinks need to be fed of which only one handles all data to finally persist them at RAC with half of the Flink nodes. The other sink only has to pass alerts to AM, which is a neglectable data volume considering the class imbalance ratio of DS-III, which is roughly 1000:2. This is one of the reasons why we leave out further elaborations of AM. Instead, let us concentrate on the hardware utilization of RAC not least because of its inherent pattern building functionality. On H6, RAC consists of four dedicated nodes that receive batches of FVs, which results in evenly distributed ingress traffic arriving to its public network interfaces. To inspect the vCPU consumption of RAC for this simple data storing activity, several isolated test runs without PBC are conducted. These confirm that processor loads caused by the insertion operations of Flink are not exceeding 12.39% and reach an average usage of 8.12%. This manifests low effort for the entire DB cluster. Thus, the remaining part can be used for the pattern building process and, in fact, RAC makes use of all 32 vCPUs to serve this duty. Combined with the persisting activities and the aforementioned replication process to transport latest patterns to Flink, RAC obtains a maximum vCPU utilization of 88.69% and accomplishes 81.08% on average. Yet considering the fourth RAC node (i.e. RAC-4) in isolation, it caters for a mean utilization of 89.17%, which is roughly 10% higher compared to the other three nodes. This result is hardly surprising because it hosts the implementation of PBC and, therefore, causes higher effort to dispatch queries and collect intermediate results. This fact is especially supported by the ingress and egress traffic of the dedicated interconnect interfaces where RAC-4 sends and receives considerably more data than the other three nodes. These observations are not reflected from a memory perspective as all RAC nodes roughly share the same RAM behavior starting from 12 up to 21 Gbytes until the incoming workload is processed completely. A comprehensive overview of RAC's resource consumption is depicted in Figure 10.25 (c). To finalize descriptions of the hardware utilization, we want to emphasize that using F00 in combination with H6 only takes around ten minutes to process the entire workload. This is opposed to the original capture time constituting more than six hours of real-world network traffic, which manifests a solid result for the entire HFIDS.

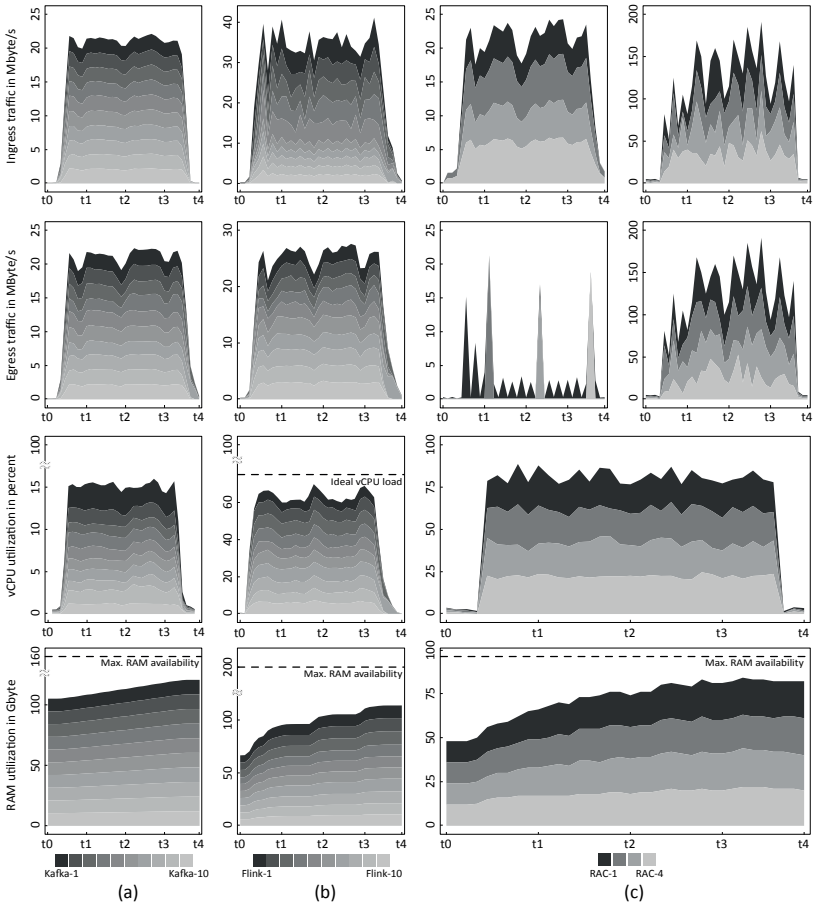

FIGURE 10.25: Cluster hardware utilization over time for F00 at H6 illustrating ingress and egress traffic, vCPU and RAM for (a) Kafka, (b) Flink and (c) RAC using stacked area charts; the two top left and the two top right plots of the RAC subcluster represent the public network interface and its interconnect respectively

In the previous paragraphs, we detailed throughput rates and corresponding speedups for all FFCs on real-world enterprise traffic and outlined the hardware utilization of the most promising F00 configuration. The remainder lifts these concrete results beyond our hardware contingent. Hence, an attempt is made to estimate such data by extrapolating earlier observations. For this examination, we use conventional regression analysis inferring two functional relationships between required hardware (x) and throughput (y), i.e. $y = f_1(x)$ describing a linear relation and $y = f_2(x)$ expressing a logarithmic behavior. The rationale behind this choice is two-fold. On the one hand, f_1 models an ideal behavior where the system is able to increase throughput rates in a linear fashion as we append more hardware. On the other hand, f_2 addresses a common problem of

FFC	Linear		Logarithmic	
	$f_1(x)$	R^2	$f_2(x)$	R^2
F00	$3141.15x + 1087.31$	99.18	$8503.33 \cdot log(x) + 2757.06$	89.17
F0W	$1332.47x + 1991.88$	97.08	$3537.58 \cdot log(x) + 2776.43$	82.95
FH0	$1548.46x + 2254.73$	98.95	$4239.14 \cdot log(x) + 3025.95$	91.55
FHW	$886.50x + 2703.85$	97.45	$2442.15 \cdot log(x) + 3128.69$	91.58

TABLE 10.10: Linear (f_1) and logarithmic function (f_2) as fitted models to extrapolate throughput rates for all FFCs; coefficient of determination R^2 is expressed in percent

FFC	Linear (f_1)				Logarithmic (f_2)			
	H10*		H20**		H10*		H20**	
	FVs/s	Speedup	FVs/s	Speedup	FVs/s	Speedup	FVs/s	Speedup
F00	32499	6.93±0.51	63910	13.63±1.25	22337	4.76±1.05	28231	6.02±1.55
F0W	15317	4.03±0.51	28641	7.54±1.25	10922	2.88±0.69	13374	3.52±1.02
FH0	17739	4.55±0.34	33224	8.52±0.84	12787	3.28±0.55	15725	4.03±0.81
FHW	11569	3.21±0.33	20434	5.67±0.82	8752	2.43±0.34	10445	2.90±0.50
*264 vCPUs and 792 Gbyte RAM, **544 vCPUs and 1632 Gbyte RAM								

TABLE 10.11: Results of the extrapolation illustrating rounded throughput rates in FVs/s and speedups including 95% confidence intervals on configuration H10 and H20 uncovered by regression model f_1 and f_2

parallel processing particularly in distributed environments, which diminishes expected speedups. These situations are a typical result of increasing communication overhead between involved computing nodes such that the time taken to share intermediate results overwhelms the effective processing time. This phenomenon is called "parallel slowdown" and, therefore, states a rather pessimistic scaling performance as opposed to f_1. Hence, analyzing both scenarios provides an intuition of how our HFIDS might behave when equipped with more hardware resources[14]. The result of this activity are presented in Table 10.10 showcasing the best fitting regression models f_1 and f_2 for each FFC. Applying them yields the same tendencies as gathered between H1 and H6, i.e. F00 obtains the most tempting throughput rates followed by FH0, F0W and FHW either using f_1 or f_2. Considering H10 and H20 as consecutive HWC, F00 reaches a speedup of 6.93 and 13.63 respectively employing f_1. This way, it is estimated that the HFIDS is capable to process 63910 FVs/s at H20 where each FV only consists of information purely derived from a single IPFIX flow. Taking into account full-featured FVs, represented by FHW, only permits a throughput rate of 20434 FVs/s using the same hardware setup, which is roughly 3.13 times lower. The other two variants F0W and FH0 obtain a throughput of 28641 and 33224 FVs/s at H20. Comparing these results with the logarithmic regression analysis reveals that throughputs received by f_2 are approximately two times weaker as numbers of f_1 at H20. As such, the increase of processable FVs/s for all FFCs just improves by 36.82% on average as opposed to our original measurements at H6. The best result in this respect is received by F00 with 28321 FVs/s while the poorest outcome is produced by FHW catering for 10445 FVs/s. This documents that F00 is 2.70 times faster compared to FHW at H20 even when speedups are not ideal. The outcome of F0W and FH0 is in between with 13374 and 15725 FVs/s. Further results of this extrapolation are highlighted in Table 10.11.

[14] Note, this estimation is based on the assumption that hardware demands continuously follow the sequence of our introduced cluster lineup given in Table 10.8.

10.3.6 Discussion

When evaluating NIDSs, an important ambition is to test the solution of interest in most realistic situations to get insights about credibility aspects ultimately. These include the answering of key questions such as: How is the attack coverage of the system and which are missed? Is the proportion of raised false alarms acceptable? To which degree are operators supported by the system to contain potential intrusions? Is the solution invariant to changes of the underlying network system and, if not, what are supplied technical remedies? Is the system capable to serve increasing throughputs demands and how can this be accomplished? In this Section 10.3, we made an attempt to address these and other related questions employing a systematic approach. First, we rendered synthetic scenarios introducing drifts with different intensities and class imbalance in order to find stable operating points. This was imperative given the absence of adequate public network traces comprising drift. Moreover, we analyzed the benefit when being guided by a small portion of GT during the live phase. We also employed another larger network trace tapped at enterprise facilities and salted it with a variety of attack instances in addition. Our intention applying this supplemental benchmark data set was three-fold. We were eager to determine if found operating points under drift also hold in real-world environments and tested other state-of-the-art learning algorithms to provide a comparative study. After all, we used the data set to stress test our HFIDS figuring out whether increasing data volumes can be compensated as more hardware resources are allocated.

The outcome applying this methodology demonstrated that the identified operating points basically work out under both synthetic and realistic circumstances. In the synthetic case, our system recovered from predictive performance breakdowns in most instances with the support of AM. Only in two out of eight scenarios, the HFIDS was incapable to regain former strength as a result of the experienced concept drift. We relate this issue to the degree of severity comprised in these two situations, which is very tough to overcome using pure flow features. By combining flow data and derived traffic statistics instead, the system managed to sort out the problem. It should be stressed though that none of the synthetic scenarios was straightforward to solve either. Quite the contrary was the case. Neither our proposed solution nor any other ML algorithm we tested withstood these situations and was able to recover completely unless GT were made available to guide the learning task. Without GT, we could even observe that the number of anomalies identified by the HFIDS was increasing throughout each drift situation. This is an important finding because this behavior showcased that the notion to express uncertainty from ADC's standpoint essentially works. Concretely metered by the ADR measure, an increasing value signals that new phenomena in the data are detected but cannot be resolved autonomously. As a result, ADR can be employed as indicator for system operators to determine when the HFIDS actually requires guidance. In a more general context, it can also be seen as some sort of drift detector. When it comes to the classification performance on real-world data where class skew is even more severe and no AM resolution is involved intentionally, several results are noteworthy. Generally, we observed that the multiclass performance is moderate. When simplifying the underlying problem to a binary one, however, overall predictions became more reliable as some attack classes were mixed up with other malicious activities.

Hence, we recommend to use the latter setting to relax the already very tough learning situation. Turning to concrete results employing this binary setup with flow data only, the outcome of the HFIDS is ambivalent. On the one hand, it is capable to generate a relative low number of false alarms. On the other hand, it misses roughly one out of three attack instances, which can be very harmful. Diving deeper into the detection capabilities lays open that comprised brute-force and spoofing attempts can be detected with certainty whereas floodings such as DoS or DDoS attacks are at the borderline. We rate the detection of all other intrusion attempts as unacceptable because hits were partially existing but very sporadic. When incorporating traffic statistics in addition, predictive deficits could be compensated to a great extent. Attack misses reduced by more than half such that approximately six out of seven attacks could be detected. The FPR also dropped significantly and became even more acceptable. In terms of attack types, the HFIDS is quite certain by detecting brute-force attempts, spoofings, floodings and web attacks combining flow and traffic statistics. Moderate detection capabilities could be attested for botnet communication, probing or other attacks where between six and seven attack instances out of ten were correctly detected. Unacceptable misses were still apparent for conducted exploits or malware types other than botnets such that only two out of five malicious instances could be unveiled. Putting these results into perspective shows that the engineering of elaborated features is an encouraging direction for flow-based intrusion detection research. While using native flow features, moderate to proficient detection capabilities could only be achieved on three out of eight attack types, we extended this characteristic to four more malicious classes by deriving additional features from pure flows data. This is a very appealing outcome towards general-purpose intrusion detection where flow technology is employed as baseline. Yet, it also becomes clear that further research is required to eradicate performance gaps particularly for those borderline attacks given by ransomware and exploit instances.

Moving to the comparative study of our HFIDS against others applied on the real-world data set salted with intrusions, the selection of competitive solutions for this subject were not straightforward. Most available approaches designed for nonstationary environments typically follow one out of two premises: (i) they either assume GT data to be fully available or (ii) they deal with limited gradual drift exclusively when operating under EVL (see Section 8.2). While the first setup is rather unrealistic due to prohibitive costs compiling the GT for a potentially unbounded data stream, the second basically applies but drifts that have to be addressed might be more radical and challenging in nature (see Section 5.2). As a result, competing methods of both mentioned directions would be simply inappropriate when deployed in our network security context due to gross simplifications or limitations mentioned. Famous IDS solutions such as Snort are applicable. The problem, though, is the difficulty to map conventional performance metrics to the outcome of such a system. Considering brute-force attacks for instance, Snort rules covering such malicious activities typically rely on counting mechanisms and raise alerts periodically. Porting this behavior to our detailed metrics are disadvantageous for Snort because a high amount of attacks would be, in fact, missed (see Section 6.4.2). From this perspective, it becomes apparent that a direct comparison among the different detection concepts is cumbersome and even harmonizing performance counters would introduce bias either way. Regarding existing flow-based IDSs, many of them can get over the mentioned performance measurement dilemma but are strictly designed

for one or very few attack types (see Section 7.2). This in turn conflicts with our ambition aiming for general-purpose intrusion detection and so these specific IDSs would fail to detect a broader range of different attack instances that are covered by our solution. Despite all of these issues, we managed to conduct a mild comparison contrasting our HFIDS with several incremental learning approaches designed to operate on data streams and, thus, on potential concept drift situations. We consider this comparison to be "mild" because obtained results should be reviewed with the perception in mind that competing approaches have access to the underlying GT right after processing unseen FVs. On the contrary, our system runs under EVL without AM involvement. This way, significantly stricter conditions are assumed in our case making the comparison clearly uneven. Yet, applying all systems to the mentioned benchmark data set showcased that our system provided an overall better predictive performance considering either a multiclass or a binary classification problem using flow data. If minor supervision would be added in our case, we are confident that these results could be even boosted. At least one could expect to bring down the *ADR* and maintain the same classification result. If traffic statistics are also incorporated, results became closer. Even though the recall of our system was slightly better than the ones of its contenders, two other methods threw noticeable fewer false alarms such that the HFIDS was finally in second position but not far off with a low *ADR* as well. With these results, we can conclude that our system is capable to compete with other algorithms that rely on complete GT available whereas our solution is permitted to be trained on these data only during the warm-up. Trying to equalize the learning setup of all algorithms to EVL revealed that the HFIDS outpaces its contenders by a big margin on all FFCs. This demonstrates that our continuous learning attempt in the live phase is beneficial particularly in terms of conducted hit rates.

We also addressed the scalability potentials of the HFIDS that are rarely pointed out by competing approaches. Using the real-world enterprise data set reflecting more than six hours of network traffic, we penetrated our solution with constantly high workloads under different HWCs and verified achievable throughput rates per FFC. In the most basic configuration with 28 vCPUs and 84 Gbyte RAM, our system was capable to get throughputs roughly between 3600 and 4600 FVs/s dependent on the FFC. Speedup behaviors were basically linear but with different gradients. The most efficient processing could be reached when employing pure flow data. This is not surprising because no dependencies between FVs existed. This way, the highest degree of parallelism could be applied exploiting operator chaining catering for a speedup of 4.31 and so the processing of the workload took only around ten minutes using a cumulated number of 152 vCPUs and 456 Gbyte RAM. The weakest outcome was obtained when involving correlations and window computations for traffic statistics apart from pure flow data (see full-featured FVs) yielding an average speedup of 2.17. These results demonstrated that the HFIDS is basically able to handle all traffic peaks of the original data volume with ease. This can be justified along the observation that the highest burst in the data were around 5000 biflows per second (see Figure 10.14) whereas our architecture was capable to process a constant data volume of approximately 7800 FV/s in worst case within the given hardware bounds. We even figured out that these hardware demands can be relaxed to some extent as some VM profiles were oversized. Despite this solid outcome, it should be stressed that throughputs are heavily influenced by correlations

and window computations. As such, they roughly drop by half either involving host data correlation or traffic statistics acknowledging a trade-off between classification and speed performance. To extend this analysis beyond available hardware resources, we also made an attempt to extrapolate given results. Assuming a linear growth of the speedups, which fits earlier behaviors best, the HFIDS was capable to process between 20434 and 63910 FV/s using 544 vCPUs and 1632 Gbyte RAM. In a more conservative setting where logarithmic characteristics were presumed, our solution could handle between 10445 and 28231 FV/s with the same hardware. Trying to give an impression what these numbers actually mean with respect to the sizing of a network, let us assume that one computer system in the network produces one biflow per second[15] on average that is sent to the HFIDS. Applying this estimate, it can be derived that our solution is able to monitor network systems consisting of between 7800 and 20100 individual machines dependent on the selected FFC within our test facilities. In case of the pessimistic extrapolation, this means that the HFIDS still can handle networks of between 10400 and 28200 computer systems even when considering parallel slowdown. A drawback of this assessment was the unavailability of real host events. As such, we could just simulate the correlation with respective Flink operations but neglected effects on model building and classification. Moreover, the evaluation of failover functionality was left out due to additional hardware demands that could not be covered by our hardware contingent.

All of these discussed results should also be contextualized in terms of operational aspects as a NIDS should be able to provide technical support to facilitate maintenance activities and follow-up assessments for raised alerts. The first point to stress in this respect is that our architecture is equipped with the earlier mentioned resolution process at AM that is closely related to the field of active learning. As opposed to other conventional active learners, the key benefit of our proposed approach is two-fold though: (i) we invert the control flow in the sense that a human expert or teacher is not required to immediately take action when the machine requests it. Instead, experts decide when to initiate the resolution process comprising a more convenient setting in practice. (ii) Once the resolution is triggered, the system delivers a list of all current anomalous clusters it is uncertain about represented by their mean. These are then expected to be labeled by operational staff and finally incorporated into the learning process. This approach demands significantly fewer human intervention compared to conventional active learning techniques because they usually rely on the labeling of individual instances. Besides this facilitation of maintenance actions provided during runtime, another central asset of our HFIDS is transparency to close semantic gaps. As system operators are obviously unwilling to rely on unexplainable alerts (e.g. [14, 15]), our system attempts to continuously extract human readable patterns from classified FVs that are injected into the detection process. This undertaking is beyond misuse detection because our method is not restricted to build patterns for malicious activities. Results show that under heavy drift, abstaining of patterns increases as predictions become more and more vague unless moderate supervision is supplied by AM. Under stable conditions, however, the HFIDS provides an explanation in at least 92% of the cases on average. This achievement is

[15] This is a very optimistic valuation as we experienced 0.30 biflows per second on average for one machine and 0.80 biflows per second for only a small portion of heavy talkers under realistic heterogeneous enterprise conditions.

in contrast to other available approaches with similar predictive qualities because most rely on black box models. Certainty, some of them use DTs and so they offer a kind of transparency. Yet, extracting the root causes for made decisions from such models can be considered a cumbersome task as the entire tree has to be inspected. In our case, though, there is a very high chance that immediate explanations are delivered by decision rules to initiate follow-up actions for raised alerts or to keep track of the system's behavior in general.

Lastly, we want to give a brief discussion on some exposed operating points and the identification process of them. Additionally, we propose some improvements for our deployment as well as outline drawbacks. Our main intention for the identification process was driven by the amount of parameters to be preset and thereby adjusting the HFIDS under different drift situations that can arise in practice. The problem behind this ambition is that drift facets can vary widely particularly for gradual changes ranging from hours to days or even weeks when considering a planned long running rollout process for new software components in the underlying network system, for instance. Thus, it becomes questionable whether operating points can be found at all for such a generic setup. This is a crucial point because long-term studies in this direction are not existent for our concrete purpose requiring further research effort. In our case, we only tested two drift variants leaving room for improvements. Nonetheless, obtained operating points worked out in both considered drift scenarios as well as under real enterprise traffic. With respect to specific parameter values, a remarkable tendency could be obtained for the pattern aging. Results showed that the best predictive outcome is reached when keeping attack patterns over 35% longer in PDB compared to benign and anomalous patterns. In fact, we experienced that attack patterns, once established, are invariant to data changes in many situations. Still, the identification process suggested a retirement anyway after retaining them for a long period. This suggestion has two origins. The first cause can be related to the data and the base implementation of PBC. If an established attack pattern with high purity becomes idle as the attack it covers is no longer reflected in the data, PBC simply drops it as part of its aging model trying to keep model complexity low. The second cause is due to class imbalance where essentially any evidence for a malicious activity in the network must be taken serious to create an attack pattern. Yet, it cannot be guaranteed that the built pattern is correct due to *FP* and so a falsely created pattern is in place that needs to be retired eventually. Hence, if we could relax both occasions, we are convinced that the identification process would advocate a much higher aging for attack patterns as the proposed ones. In terms of improvements and drawbacks, several aspects come to mind which we outline shortly. The first point we want to discuss is related to the resolution process at AM. In the current configuration, all anomalous clusters need to be labeled by operational staff. On the one hand, this could be refined if the HFIDS would provide a recommendation for the true class dependent to the data distribution. On the other hand, the system could prioritize the list of anomalous clusters with approximated impacts on the classification once labeled by an operator. This way, manual engagement can be optimized further. The second point is also connected to the resolution of anomalous clusters but more subtle in nature. By analogy to the cluster assumption of semi-supervised learning (see Section 2.2.1), we also adopted this premise during our experiments where anomalous clusters were labeled according to the most plausible class. This assumption, however,

might not always hold true necessarily. This signifies that our results could have been better when labeling concrete FVs instead of clusters abstracting them. On the flip side, this would mean significant more manual intervention. Thus, a more promising direction to deal with this affair is to perform a deeper cluster analysis based on flows in order to anticipate clusters associated to multiple classes. Next, we look at enhancements for ADC. Its offline component currently takes care to clear outdated flows from TDB. This functionality could be taken over by RAC completely using "in-DB archiving" where processed data can be compressed and made invisible. This must be aligned with AM as TDB is also a valuable source to render reports in order to keep track of the system behavior. These reports are not implemented so far but would be an expedient supplement. Another interesting point in terms of in-DB processing would be to bring the offline component of ADC to RAC. This certainly relaxes i/o at TDB as fewer external reads and writes are issued against that table. A positive side effect of this undertaking is that ADC can become a scale-out solution eventually when ported to RAC. A further improvement would be to use in-memory DB technology, which can boost performance significantly because we can get rid of disk-based access patterns at RAC. This might also have implications to the complexity of the HFIDS as deployment delays of produced patterns between PDB and its local counterparts might be eliminated. The problem, though, is that a seriously higher financial expenditure can be expected that should not be underestimated because capacities at TDB need to be covered and a high-speed network need to be engaged in order to compensate drawbacks of shared-nothing architectures most scale-out in-memory solutions are based on. Another problem is not directly related to in-memory technology but to the sizing of ADC's ensemble. It is fixed to five members and increasing this number, which would be beneficial in terms of predictive performances, is not straightforward from a technical standpoint. The underlying reason rests upon the current implementation in Flink. It is sequential rather than parallel and so its elasticity is currently bounded from that perspective. Lastly, we want to stress that InDBR as base implementation of PBC relies on discretized data. Thus, a next consequent step would be its refinement to continuous data. This implies the integration of further arithmetic operators increasing the expressiveness of patterns. Another benefit from this undertaking is that some actions in DPC can be omitted. It is also a fruitful attempt to further enhance the explanatory power of patterns by weighting literals, for instance. With such meta data, system operators could get an intuition about the importance of different parts in patterns and their contribution to the overall decision-making process.

10.4 Summary

In this chapter, we brought together ideas and concepts elaborated throughout this work in order to assemble and deploy a new HFIDS eventually. The result is an architecture that technically relies on a computer cluster with three subclusters (see Kafka, Flink and RAC) and offline components. While the message queue Kafka intermediately buffers incoming IPFIX flows and optional host events from the network system to safeguard, Flink consumes this input in a stream-like fashion and serves as MIP. In this pipeline, data undergo preprocessing activities dependent on the selected FFC mode to create

FVs that are inspected by PMC and the optional ADC. Afterwards, processed FVs are persisted in TDB with their classification result and alerts are sent out to AM concurrently in case a malicious FV has been observed. The DB system RAC implements TDB and the pattern building at PBC, which are in turn deployed at PDB. To this point, the system realization mainly corresponds to the stipulated operational sequence postulated in Chapter 5. Yet, several important aspects required technical refinements. We tailored ADC in such a way that its functionality is split in an online and offline component. While its online component takes care of the conventional classification inside MIP, its offline counterpart monitors the online performance and updates weights or the entire model if necessary. Additionally, the offline component is in charge of clustering the data to anticipate new structures. These have to be resolved by AM providing the true class for such clusters, which are finally looped back into the learning cycle once clarified. Before ADC can be turned operational, a warm-up phase is introduced, which basically corresponds to the initial training where detection mechanisms are aligned with the underlying network infrastructure to observe. We also made minor modifications to PBC and window definitions as part of the preprocessing in MIP. In case of PBC, new states were introduced for entries at TDB and the deletion of outdated FVs was delegated to ADC's offline component instead. For those windows to compute traffic statistics, we refined their scope in order to keep track of most recent data only. Lastly, access patterns between PMC and PDB required revision because of discovered speed discrepancies. These could be sorted out by employing a cache mechanism, which distributes created patterns in PDB to individual Flink nodes in an incremental fashion justifying the overall architecture technically.

Apart from these technical considerations, we also carried out extensive experiments in order to outline the practical characteristics of the HFIDS. In synthetic scenarios, the system could recover from most drifts with AM assistance. The application to realistic enterprise data disclosed also promising results where human intervention was neglected by purpose. Even though we can conclude that general-purpose intrusion detection is difficult to conduct using native flow data as many attack types are simply missed, the incorporation of traffic statistics certainly payed off. This outcome basically confirms the findings of Chapter 7. The distinct difference, though, can be attributed to the learning scenarios employed herein. They can be considered much more challenging and realistic at the same time covering class imbalance, multiclass classification and concept drift all embraced by an incremental learning attempt and EVL. In this setup, our system essentially was able to detect seven out of eight attack classes and out of these seven classes four were very convincing. In addition, produced false alarms were also acceptably low with or without traffic statistics. Despite of difficulties to relate our results with other approaches, a comparative study could be compiled. It showed that the HFIDS is good at handling pure flow data in contrast to other state-of-the-art incremental learning methods. When utilizing traffic statistics as well, result were closer. However, we should emphasize that our system only saw GT data during the initial warm-up whereas others had access to associated classes throughout the benchmarks. If learning settings were harmonized towards EVL, the HFIDS won by a big margin taking advantage of its continuous adaption. Another benefit of our solution is transparency as it provides an explanation for nine out of ten predictions in the form of patterns. In terms of scalability, results revealed that the HFIDS is capable to countervail growing

workloads by appending more computing nodes to each subcluster. Employing the highest resource configuration in our test facilities, we demonstrated that the system can process at least 7800 FVs/s consistently, which is beyond original enterprise workload peaks experienced. Even in a pessimistic extrapolation, the system was estimated to handle 10400 FV/s. This is a noteworthy outcome for a centralized NIDS. Still, it should be emphasized that speedup characteristics are heavily influenced by window computations and so one has to find a balance between classification capabilities and expected data throughput when hardware resources are scarce.

Part IV

Epilogue

Chapter 11

Conclusion and Outlook

This work provided insights into the multifaceted nature of network security and addressed several critical aspects of it by outlining a concrete opportunity to efficiently monitor private enterprise and institution network systems that are expedient playgrounds for attackers. In this chapter, we retrospect on pathological findings as well as on encouraging results that were disclosed during our research endeavor. In addition, an outlook is presented to potential improvements and new directions to safeguard local and joint assets against theft, abuse and further damages caused by the pervasive threat landscape.

11.1 Conclusion

This thesis strove for the design and the development of a novel HFIDS, a concrete hybrid flow-based intrusion detection solution. It was built on top of five key pillars, i.e. adaptivity, scalability, transparency, flow monitoring and reliability, justifying the main contribution of this work from a broader perspective. We conclude on this solution herein by revisiting each of its pillars successively.

During the course of this work, we argued repeatedly that the problem to monitor and safeguard network systems is not trivial. This is clearly evident when reviewing the diverse threat landscape a NIDS has to face nowadays. Another decisive aspect that heats up the complexity of our problem domain is that network systems are not static per se. Dynamic user behavior, software updates or entire segment extensions can change the overall footprint of a network crucially. Apart from these legitimate transformations, new attacks can emerge or characteristics of existing attacks are changed intentionally to evade detection. From a ML perspective, such nonstationary phenomena known under the term "concept drift" render uncertainty causing learning performances to degrade. Although neglected in network security literature oftentimes, our proposed solution anticipates these dynamics through its adaptivity pillar. In situations where labeled data are rare and only available during an initial warm-up phase (see EVL setting), results revealed that the HFIDS is capable to handle slow incremental drifts self-sufficiently. Gradual and sudden drifts in turn could not be handled by the system directly because their radical nature prevented a reasonable tracking in an automated fashion. Yet, we were able to identify the changes induced by those drifts objectively. This became possible by introducing a mechanism as part of the system's inherent anomaly detector ADC requiring manual resolution. In this respect, we developed a process supporting operational staff to resolve gradual and sudden drifts conveniently. We term it convenient because, unlike conventional methods, concrete intervention is triggered by humans rather than by the machine. Moreover resolutions are not required to be performed on individual network activities but on summaries of them reducing the amount of data to be inspected considerably. Lastly, a numeric measure was devised as global indicator for uncertainty guiding system operators when to step in. Based on this delivered functionality to facilitate maintenance activities, we conclude unfavorable manual intervention still to be tolerable given the severity of those drifts. Further moving to another dynamic characteristic of network systems, underlying hardware upgrades can have immense implications for any NIDS particularly when link speeds are technically upgraded. As a result, the velocity of traffic to be monitored increases as well, which can overwhelm initially presumed processing capabilities of security solutions easily. To compensate such undesirable circumstances provoking potential data loss, the scalability is of paramount importance. Our proposed HFIDS was intrinsically designed around this pillar. Within our test environment, we demonstrated that growing workloads can be addressed by appending more hardware resources. Using all available hardware in our testbed, achieved data throughputs were consistently high and even a pessimistic extrapolation estimated our system proficient enough to monitor medium to large production networks. This outcome can be considered sound given that our HFIDS is a centralized NIDS. However, we also realized that enriching network activities with further context impacted the speedup of the system negatively.

The third pillar of the HFIDS is concerned with transparency to close semantic gaps. This is a critical factor for the overall acceptance of the system. To put this pillar into effect, we engaged PBC, a new rule-based algorithm that incrementally learns human readable patterns from incoming data, which stems from a general development that is also part of this work. Two ingredients were key for its seamless integration into our architecture. On the one hand, there was the demand to deploy this algorithm inside DB facilities in order to resolve and harmonize different technical aspects of the system with the ultimate goal to reduce communication overhead. On the other hand, appropriate handling of uncertainty was required, which is not directly related to concept drift as argued earlier but to information deficiencies caused by the imperfect nature of input data expected to be processed (see next paragraph). We, therefore, worked out an in-DB rough set model as groundwork designed for both affairs. It has a superior worst case runtime than conventional rough set implementations and produces low communication overhead. Compared to closer approaches using same technologies, theoretic upper bounds are identical but competitors are not fully compliant with rough sets in the proper sense and so they cannot manage vagueness as opposed to our model. In addition, our in-DB implementation also features variable precision. Employing this framework for PBC permitted to generate certain and uncertain decision rules by using a novel bottom-up induction strategy, which has a fast turnaround in terms of converting input data into concrete patterns. In that sense, certain rules are utilized for classifications and explanations while relaxing ADC at once. Uncertain rules are too inaccurate for this purpose but they pose an interesting prospect for future situations as data evolve over time. Even though this constellation is reactive, meaning that new patterns are successively rendered after an unexplainable classification at ADC, results were very promising. In fact, we showed during our experiments that nine out of ten predictions performed by the HFIDS were initiated by patterns rather than by ADC. This effect can be attributed to the constant generalizing attempt of PBC that increases the chance for a pattern match even for unseen data. Under heavy drift, however, more and more patterns are becoming vague and as a result the level of transparency drops as well unless moderate supervision is supplied (see previous paragraph). Although this outcome is generally poor, it demonstrates that our uncertainty management pays off because the system should only provide explainable predictions if it is most certain about them. Thus, we conclude our solution to be a valuable contribution for explainable ML within the problem domain of network security in principle.

Given that our HFIDS is intended to consume network activities in the form of IPFIX biflows, special attention was paid to the subject of selecting the right features separating benign and malicious network footprints sufficiently well. This undertaking, which is at the heart of both the flow monitoring and the reliability pillar, was crucial because the universal effectiveness of flow technology for intrusion detection has not been fully studied in literature. As a result of our analysis, several minimal feature subsets were identified. To assess their credibility for both classic and modern attacks, we applied those feature combinations together with a bunch of different base ML algorithms on our newly compiled benchmark data set NDSec-1 as well as on further independent network traces. For the most part, the outcome disclosed that flow features produce solid predictive performances taking into account the very basic nature of employed learners during the evaluation. We even could improve results by incorporating traffic statistics

across adjacent flows or security-related log information residing on host level that has to be correlated with flows. Yet, this flow feature analysis considered class balance and static data distributions leading to an idealized test environment. We tightened these aspects when evaluating the final HFIDS by taking into account class skew, concept drift and EVL after all. Within this realistic and surely more challenging scope, results on pure flow data evinced few false alarms and documented solid detection of some noisy attacks whereas the covering of most other attacks were very sporadic. When utilizing traffic statistics in addition, classification performances could be improved to a great extent such that detection capabilities on all attack types could be demonstrated with the exception of specific malware and exploits. However, these improvements were too elementary and so general-purpose characteristics cannot be attested entirely for our proposed solution unless noisy attack types such as brute-force, spoofing, flooding or web attacks are considered exclusively in a binary setting. This leads to the conclusion that flow-based intrusion detection is not a panacea for network security at least from the present state of research. One has to realize that the deployment of a NIDS with low installation and maintenance costs is enabled by flow technology but a reduced attack coverage must be accepted as well. If the monitoring of a broader attack spectrum is desired, one has to revert to traditional full packet inspection as it relies on a much finer data basis as opposed to flows. On the flip side, significant higher expenditures must be taken into account to run such a security solution in practice. But even accepting these implications, state-of-the-art packet inspection is still debatable because it is prone to encrypted communication, which is becoming the norm rather than the exception. Having observed this trend over the last years for legitimate communication, network crime scenes are also increasingly disguised by attackers using encryption techniques. Combined, this situations slowly destabilizes the benefit of packet-based inspection except when intentional decryption backdoors or proxies are established radically weakening privacy aspects. This puts flow technology into the spotlight again since it is less susceptible to encryption. As such, we want to emphasize that our conclusion on general-purpose flow-based intrusion detection is not discouraging or dismantling the research endeavor in this work by any means. Quite the opposite is the case. Our constant efforts are rather underpinned as enhancing and contextualizing flows with other information increased the range of detectable attacks paired with a reliable false alarm performance under very realistic circumstances. From this perspective, our attempt was certainty a step in the right direction not least because related flow-based intrusion detectors largely concentrate on isolated attacks only whereas our results document a broader attack coverage that can be grasped as a kind of baseline protection.

11.2 Outlook

As part of the system deployment and evaluation, we already discussed some functional aspects that might improve our proposed hybrid architecture. Herein, an outlook is presented on several other notable connecting factors that are worth mentioning giving rise to ample research.

Based on the outcome of this work, we concluded that our HFIDS, in the present version, can only cover parts of the current threat landscape convincingly. Yet, we also

postulated our strong belief, buttressed by positive empirical results, that the potentials of flow-based intrusion detection are still not exhausted reducing miss rates and false alarms considerably. One direction we see to accomplish this mission is to seek for further valuable information extractable from traffic statistics and security-related log information. In this respect, one could also investigate opportunities to which degree underlying computations can be brought closer to the actual flow metering and exporting of the IPFIX protocol. On the one hand, this optimization has the obvious benefit to get rid of cumbersome preprocessing activities inside the HFIDS. On the other hand, integrating the computation of statistics and correlations into the central flow creation process might lead to a more efficient processing as data are managed in one go. At this point, we should also mention that a huge amount of features officially defined by the IPFIX standard (see IEs) actually remained unexplored. This is due to the flow exporter employed in this work whose flow feature catalog only supports a subset of available IEs even though proclaimed to be the reference implementation of IPFIX. This in turn opens another promising direction to improve performances towards general-purpose characteristics by including residual IEs as well. These could also be combined with those valuable features found in this work to consolidate our flow feature analysis. If this information is still insufficient, one could also elaborate on new IEs as IPFIX literally supports information of any layer in the TCP/IP stack.

Further points, worth to look into, would be failover scenarios, which were out-of-scope in this work. In this regard, a particular focus should be on the subsystems Kafka and Flink serving data ingestion and main processing pipeline of the HFIDS in order to gain practical insights to hardware demands required to obtain seamless operations. A second point in this direction is whether scalability potentials discovered as part of our experiments are still maintainable during normal processing, outages or recoveries. Closely connected to failover support are resilience aspects within a malicious context where the HFIDS itself becomes a site of cybercrime. Some of such resilience scenarios indeed can be handled by standard recovery functionality, but there are more severe and subtle cases too. For instance, an attacker could try to destabilize the trustworthiness of our solution indirectly by injecting random content to the network landscape in the hope to cause confusion, which might generate potential false alarms eventually. Another concrete but more systematic tactic with similar effects would be to target vulnerabilities of the IPFIX metering process directly. For example, let us consider an invader in the position to flood and exhaust the flow cache. According to protocol, the metering process would be forced to release flows from the cache earlier than usual, which would lead to the exporting of corrupt flows consisting of nothing more but one packet in the most extreme case. Situations could get even more serious when intruders already compromised system components of the HFIDS permitting to perform any kind of damage from the inside. Thus, it is essential to develop countermeasures that have to be deeply anchored in the HFIDS. If it is not possible to preclude such affairs completely, developed resilience methods should at least provide some sort of guarantee keeping the system fit for purpose even after a successful take over of one or several computing nodes.

When looking at the architecture of our HFIDS in principle, a next consequent step could be its expansion from an centralized to a decentralized intrusion detection solu-

tion. The motivation for this undertaking is similar to the deepened analysis of flow features mentioned earlier in this section but, unlike previous argumentation, a whole new spirit comes into play enhancing opportunities due to a larger operating range. In this light, our vision relies on multiple HFIDSs located on different network sites that are interconnected via a global overlay network serving as communication channel, which permits to build a collaborative IDS ultimately. An enabler for such a solution is certainty the inherent transparency pillar of each local HFIDS rendering interpretable and modular knowledge that is sharable. As such, knowledge about a particular cyberthreat gathered at one site can be made available to other collaborating sites and vice versa. Due to the federation, synergies can be even extended towards the detection of large-scale coordinated attacks, which would be very difficult to achieve in the context of a single site alone. The flip side of this rewarding idea is that cross-site affiliations are not homogeneous in the general case and so there is a challenge largely impacting this endeavor, i.e. privacy. Therefore, new mechanisms must be devised to manage information and knowledge disseminations according to local and global privacy regulations. In [48], we made a first suggestion in this direction along with further important requirements including resilience. This discussed expansion of the HFIDS is of particular interest for emerging applications associated with the "industrial Internet". Within this scope, enterprises are expected to reach an unprecedented level of collaboration such that individual IT infrastructures literally conflate to a unified system. Thus, there are huge incentives on all participating sides to protect the assets of that holistic system collectively.

Lastly, it should be reemphasized that building quality benchmark data sets and making them publicly available is the "bread-and-butter" for network security given its data-driven nature. This conception is increasingly picked up by the research community and evident through recently compiled data sets such as CICIDS-17 and the CIDDS family apart from our effort introducing the NDSec-1 network traces. Consequently, future work on our HFIDS should involve empirical studies with those newer reference data as well in order to extend our findings. However, we cannot stop there as several aspects carved out during this work including those addressed in this section suggest that the subject of appropriate benchmark data needs to be revisited. For instance, conducted experiments did not exceed interrelated network traces with realistic data volumes greater than one working day. This is critical especially when envisioning the long-term behavior of real-world network landscapes along with their dynamics. In fact, we are not aware of any contemporary and relevant data set serving this matter. Moreover, reference data are missing to assess "advanced persistent threats" or to explore collaborative intrusion detection scenarios thoroughly. These concrete points not only document notable gaps of network security but also disclose imperative and exciting research journeys to follow.

Appendixes

A.1 Flow Feature Catalog

The following Table A.1 is an extractable composition of flow attributes provided by the IPFIX exporter YAF and represents the foundation of features for this work. In particular, it is utilized in Chapter 7 as baseline for the flow feature analysis. Note that IEs annotated with superscript * stand for PEN 6871 and those with ** for PEN 29305 respectively.

IE ID	IE name	Description
-	-	The name of the collector that received the flow.
-	-	Flow key hash of the five tuple
152	flowStartMilliseconds	Flow start time in ISO 8601 format with milliseconds
153	flowEndMilliseconds	Flow end time in ISO 8601 format with milliseconds
161	flowDurationMilliseconds	Flow duration in milliseconds
-	roundTripTime	Round-trip time estimate in fractional seconds
56	sourceMacAddress	Source MAC address of the first packet in the forward direction of the flow
80	destinationMacAddress	Destination MAC address of the first packet in the reverse direction of the flow
8	sourceIPv4Address	IPv4 address of flow source or biflow initiator present for IPv4 flows without IPv6-mapped addresses only
12	destinationIPv4Address	IPv4 address of flow source or biflow responder present for IPv4 flows without IPv6-mapped addresses only
27	sourceIPv6Address	IPv6 address of flow source or biflow initiator present for IPv6 flows or IPv6-mapped IPv4 flows only
28	destinationIPv6Address	IPv6 address of flow source or biflow responder present for IPv6 flows or IPv6-mapped IPv4 flows only
7	sourceTransportPort	TCP or UDP port on the flow source or biflow initiator endpoint
11	destinationTransportPort	TCP or UDP port on the flow destination or biflow responder endpoint
4	protocolIdentifier	IP protocol of the flow
86	packetTotalCount	Number of packets in forward direction of flow
86**	reversePacketTotalCount	Number of packets in reverse direction of flow
85	octetTotalCount	Number of octets in packets in forward direction of flow
85**	reverseOctetTotalCount	Number of octets in packets in reverse direction of flow
14*	initialTCPFlags	TCP flags of initial packet in the forward direction of the flow where each flag is represented by the first character in the flag's name: FIN, SYN, RST, PSH, ACK, URG, ECE, CWR

15*	unionTCPFlags	Union of TCP flags of all packets other than the initial packet in the forward direction of the flow where each flag is represented by the first character in the flag's name: FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
16398*	reverseInitialTCPFlags	TCP flags of initial packet in the reverse direction of the flow where each flag is represented by the first character in the flag's name: FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
16399*	reverseUnionTCPFlags	Union of TCP flags of all packets other than the initial packet in the reverse direction of the flow where each flagt is represented by the first character in the flag's name: FIN, SYN, RST, PSH, ACK, URG, ECE, CWR
40*	flowAttributes	Flow attributes in hexadecimal format
16424*	reverseFlowAttributes	Reverse Flow attributes in hexadecimal format
184	tcpSequenceNumber	Initial TCP sequence number of the forward direction of the flow
184**	reverseTcpSequence-Number	Initial TCP sequence number of the reverse direction of the flow
33*	silkAppLabel	Application label as reported by YAF using DPI
35*	payloadEntropy	Shannon entropy calculation of the forward payload data
16419*	reversePayloadEntropy	Shannon entropy calculation of the reverse payload data
136	flowEndReason	Flow end reason code: NULL=ended normally, idle=idle time elapsed, active=active time elapsed, eof=flow was still active at the end of reading
502*	dataByteCount	Total bytes transferred as payload
16886*	reverseDataByteCount	Total bytes transferred as payload in the reverse direction
503*	averageInterarrivalTime	Average number of milliseconds between packets
16887*	reverseAverageInter-arrivalTime	Average number of milliseconds between packets in reverse direction
504*	standardDeviation-InterarrivalTime	Standard deviation of the interarrival time for up to the first ten packets
16888*	reverseStandard-DeviationInter-arrivalTime	Standard deviation of the interarrival time for up to the first ten packets in the reverse direction
223 (509*)	tcpUrgTotalCount	The number of TCP packets that have the URGENT Flag set
16893*	reverseTcpUrgTotal-Count	The number of TCP packets that have the URGENT Flag set in the reverse direction
500*	smallPacketCount	The number of packets that contain at least 1 byte but less than 60 bytes of payload
16884*	reverseSmallPacketCount	The number of packets that contain at least 1 byte but less than 60 bytes of payload in reverse direciton
510*	largePacketCount	The number of packets that contain at least 220 bytes of payload
16894*	reverseLargePacketCount	The number of packets that contain at least 220 bytes of payload in the reverse direction
501*	nonEmptyPacketCount	The number of packets that contain at least 1 byte of payload
16885*	reverseNonEmpty-PacketCount	The number of packets that contain at least 1 byte of payload in reverse direction
506*	maxPacketSize	The largest payload length transferred in the flow
16890*	reverseMaxPacket-Size	The largest payload length transferred in the flow in the reverse direction
508*	standardDeviation-PayloadLength	The standard deviation of the payload length for up to the first 10 non-empty packets
16892*	reverseStandardDev-iationPayloadLength	The standard deviation of the payload length for up to the first 10 non-empty packets in the reverse direction

507*	firstEightNonEmpty-PacketDirections	Represents directionality for the first 8 non-empty packets. 0 for forward direction, 1 for reverse direction
505*	firstNonEmptyPacket-Size	Payload length of the first non-empty packet
16889*	reverseFirstNonEmpty-PacketSize	Payload length of the first non-empty packet in the reverse direction

TABLE A.1: Feature composition considered for the flow feature assessment

A.2 Learner Settings for the Flow Feature Analysis

Based on the experimental results conducted in Section 7.4 to seek for adequate classification performances on DS-1, diverse parameter settings were tested extensively. Table A.2 provides details about these parameter combinations per FFC, LC, and learner that are necessary to obtain the best outcome presented in Figure 7.3 and in those which follow in the course of that particular section.

FFC	LC	Learner	Parameters
F00	TREE	ctree	$d = 3, m = 0.99$
		C4.5	C=0.2550, M=1
		ID3	-
	RULE	C5.0-R	-
		PART	$t = 0.50$, $p = no$
		RIPPER	$n_o = 3$, $n_f = 3$, $W_m = 1$
	ANN	MLP	$L_1 = 5.00$, $L_2 = 0.00$, $L_3 = 0.00$
		MLP-WD	$L_1 = 5.00$, $L_2 = 3.00$, $L_3 = 0.00, d = 0.0010$
		RBFN	$s = 5$
	SVM	SVM-L	$c = 1.00, L = L_2$
		SVM-P	$d = 3, s = 0.10, C = 1.00$
		SVM-R	$\sigma = 0.1293$, $C = 1.00$
	MISC	HDDA	$t = 0.05$, $m = all$
		kNN	$k = 3$, $W = 10000$
		NB	$k = false$, $f_l = 0.00$
F0W	TREE	ctree	$d = 3$, $m = 0.50$
		C4.5	$C = 0.2550$, $M = 1$
		ID3	-
	RULE	C5.0-R	-
		PART	$t = 0.01$, $p = yes$
		RIPPER	$n_o = 3$, $n_f = 3$, $W_m = 1$
	ANN	MLP	$L_1 = 5.00$, $L_2 = 2.00$, $L_3 = 0.00$
		MLP-WD	$L_1 = 5.00$, $L_2 = 3.00$, $L_3 = 2.00$, $d = 0.00$
		RBFN	$s = 5$
	SVM	SVM-L	$c = 0.25$, $L = L_1$
		SVM-P	$d = 3$, $s = 0.10$, $C = 1.00$
		SVM-R	$\sigma = 0.0332$, $C = 1.00$
	MISC	HDDA	$t = 0.05$, $m = all$
		kNN	$k = 3$, $W = 10000$
		NB	$k = false$, $f_l = 0.00$

TABLE A.2: Best parameters found per learner, LC and FFC applying DS-1

A.3 Friedman Test

A brief introduction of the Friedman test is provided in this appendix. Original described in [456], it is a non-parametric test analyzing two or more data series to determine whether there is significant deviation by comparing the mean ranks of each series according to a χ^2 statistic which we describe in a moment. Before that, let us first assume an abstract experimental setup as starting point where the performance of k classifiers is measured on n selected data sets to contrast different characteristics and tendencies. Hence, the resulting tableau carrying all obtained measurements $s_{i,j}$ for each classifier c_j and data set t_i with $1 \leq j \leq k \in \mathbb{N}$ and $1 \leq i \leq n \in \mathbb{N}$ may be represented by Table A.3.

	c_1	...	c_k
t_1	$s_{1,1}$	...	$s_{1,k}$
...	...	...	...
t_n	$s_{n,1}$	...	$s_{n,k}$

TABLE A.3: Abstract experimental results of a performance analysis on classifiers applied to different data sets

As opposed to other statistical hypothesis tests, the Friedman test does not assume a normal distribution because it leverages a ranking $rank(s_{i,j})$ instead of using the pure performance scores $s_{i,j}$. In this respect, $rank(s_{i,j})$ is the fractional rank of $s_{i,j}$ according to the sequence $S_i = s_{i,1}, ..., s_{i,k}$, i.e. the best score in S_i gets the first rank, the second best score in S_i gets the second rank etc. This consideration given, we are able to compute the Friedman test statistic as follows

$$\chi_F^2 = \frac{12n}{k(k+1)} \Big(\sum_{j=1}^{k} \big(\frac{1}{n} \sum_{i=1}^{n} rank(s_{i,j}) \big)^2 - \frac{k(k+1)^2}{4} \Big) .$$

Furthermore, let be the "null hypothesis" h_0 stating that no significant distinction is observable in the series of experiments and the alternative hypothesis $h_1 = \neg h_0$. According to the test procedure, h_0 is supported if $\chi_F^2 \leq \chi_{\alpha,d_f}^2$, which means that the performances of the classifiers are insignificantly different. In all other cases, rejecting the null hypothesis becomes prevalent concluding h_1 indicating a statistical significant deviation instead. Note, the critical value χ_{α,d_f}^2 can be computed or is obtainable from statistical tables of the χ^2 distribution (e.g. [486, 487]) with respect to the "level of significance" α and the degree of freedom $d_f = k - 1$. To either confirm or deny statements made by the Friedman test, we employ the Wilcoxon signed-ranks test in addition, which is highlighted in Appendix A.4.

A.4 Wilcoxon Signed-Ranks Test

Similar to the Friedman test outlined in Appendix A.3, the Wilcoxon signed-ranks test [457] is a non-parametric statistical hypothesis test reliant on ranks. However, only two series of experiments are comparable and also the ranks are computed differently. Unlike the ranking of obtained measurements per data set, the Wilcoxon test calculates

the difference between two measures per data set and utilizes their absolute value for the ranking. To establish an example, let be an abstract data table with performance measures $s_{i,1}$ and $s_{i,2}$ for two classifiers c_1 and c_2 on data set $t_i, 1 \leq i \leq n \in \mathbb{N}$ given by Table A.4.

	c_1	c_2
t_1	$s_{1,1}$	$s_{1,2}$
...	...	...
t_n	$s_{n,1}$	$s_{n,2}$

TABLE A.4: Abstract experimental results of a performance analysis on two classifiers applied to different data sets

First, the Wilcoxon test determines those n differences $\triangle_i = s_{i,2} - s_{i,1}$ and uses them in a second step to compute the fractional rank $rank(abs(\triangle_i))$ based on the absolute value $abs(\cdot)$ according to the sequence $S = abs(\triangle_1), ..., abs(\triangle_n)$, i.e. the smallest absolute difference in S obtains the first rank, the second smallest absolute difference in S obtains the second rank etc. That given, the ranks are split according to their sign into two sums W_+ and W_- defined by

$$W_+ = \sum_{\triangle_i > 0} rank(abs(\triangle_i)) + \frac{1}{2} \sum_{\triangle_i = 0} rank(abs(\triangle_i))$$

and

$$W_- = \sum_{\triangle_i < 0} rank(abs(\triangle_i)) + \frac{1}{2} \sum_{\triangle_i = 0} rank(abs(\triangle_i)) \ ,$$

which means that W_+ is the sum of ranks where c_2 outruns c_1 and likewise the same is expressed by W_- for c_1. Moreover, W_+ and W_- also contain an additional term $1/2 \sum_{\triangle_i = 0} rank(abs(\triangle_i))$ to equalize potential ties among the two base sums. The test procedure accepts the null hypothesis h_0 (i.e. no difference can be identified among the data series) in cases where $\min(W_+, W_-) \geq w_{\alpha,n}$ or concludes $h_1 = \neg h_0$ otherwise. In this respect, the critical value $w_{\alpha,n}$ is obtainable from statistical lookup tables for this particular test according to significance level α and n (e.g. [486, 487]).

A.5 Supplemental Plots to Find Operating Points

Towards the quantification of operating points for ADC and PBC in Section 10.3.2, we identified adequate values for member weights, confidence thresholds and agings resting upon a condensed outline per FFC (see Figure 10.18 and 10.19 in particular). These aggregations are a result of fitting the grid search outcome via interpolations for each synthetic drift data set. The following two figures show the baseline for the conducted fusion per FFC, i.e. Figure A.1 for w_{min}, ψ_{min} and Figure A.2 for α_c where c is one of the classes "benign", "malicious" or the meta class "anomalous". To obtain Figure A.1, we used the cluster radius 0.16 for F00 and 0.04 for F0W, $\alpha_\perp = 1$ and a train/test ration of 4 : 1, while $\beta = 0, t = 1, w_\kappa = 5000$ and $|T| = 1000$ are fixed for Figure A.2. Note, we chose $g(j) = 2/5 \cdot e^{-j/4}$ for F00 and $g(j) = 3/7 \cdot e^{-9j/32}$ for F0W based on experience.

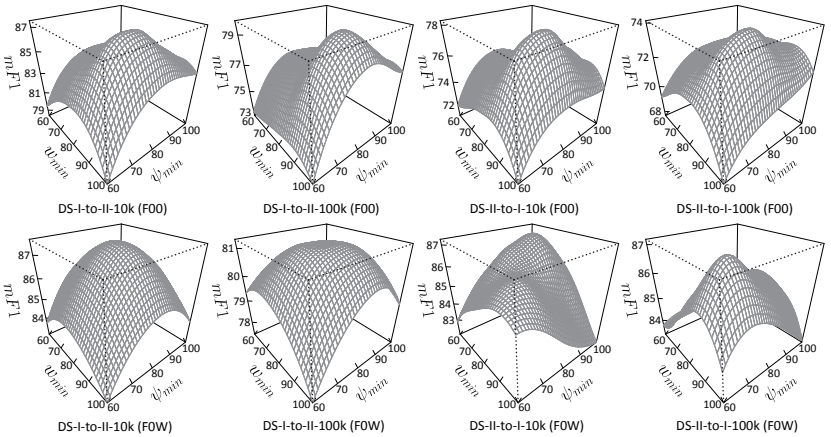

FIGURE A.1: Interpolated grid search result on all four simulated drift scenarios and FFC showcasing the average $mF1$ after three test runs with fixed cluster radius, train/test ratio and $\alpha_\perp$ to find the best operating points w_{min} and ψ_{min} for ADC's ensemble; higher $mF1$-score indicates better prediction characteristics; all axes are expressed in percent

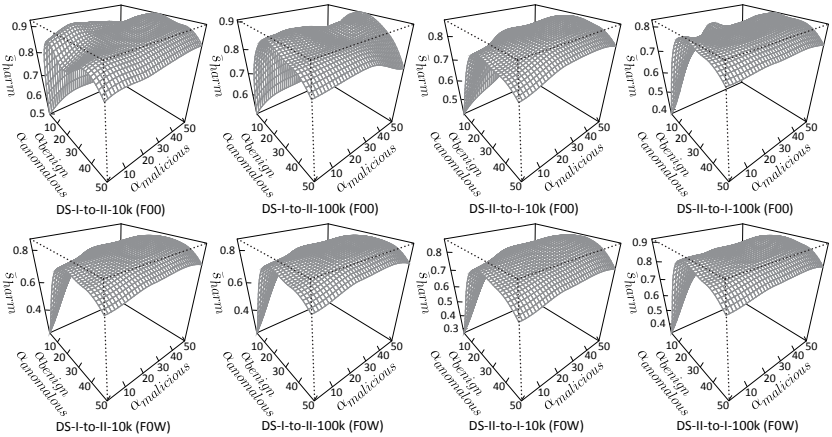

FIGURE A.2: Interpolated result of the grid search on all four simulated drift scenarios and FFC showcasing the average $\bar{s}_{harm}$ after three test runs with fixed $\beta, t, w_\kappa, |T|$ and $g(j)$ to find the best aging α_c for PBC; higher $\bar{s}_{harm}$ indicates better performance in terms of the trade-off between prediction and abstaining characteristics

References

Online references and in-text hyperlinks were last accessed on January 19, 2021.

[1] H. Dreger, A. Feldmann, V. Paxson, R. Sommer: "Operational experiences with high-volume network intrusion detection". In: B. Pfitzmann, P. Liu (eds.) *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pp. 2–11, ACM, 2004.

[2] M. Gao, K. Zhang, J. Lu: "Efficient packet matching for gigabit network intrusion detection using TCAMs". In: *20th International Conference on Advanced Information Networking and Applications (AINA 2006)*, vol. 2, pp. 249–254, IEEE, 2006.

[3] J. Kořenek, P. Kobierský: "Intrusion detection system intended for multigigabit networks". In: P. Girard, A. Kraśniewski, E. Gramatová, A. Pawlak, T. Garbolino (eds.) *2007 IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS 2007)*, pp. 361–364, IEEE, 2007.

[4] R. Hofstede: "Flow-based compromise detection", University of Twente (Twente, The Netherlands), PhD Thesis, 2016.

[5] F. Beer, U. Bühler: "Feature selection for flow-based intrusion detection using rough set theory". In: G. Fortino, M. Zhou, Z. Lukszo, A. Vasilakos, F. Basile, C. Palau, A. Liotta, M. Fanti, A. Guerrieri, A. Vinci (eds.) *Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC 2017)*, pp. 617–624, IEEE, 2017.

[6] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras: "Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX". *IEEE Communications Surveys & Tutorials*, vol. 16(4), pp. 2037–2064, IEEE, 2014.

[7] B. Claise, B. Trammell, P. Aitken: "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information", IETF, RFC 7011 (Internet Standard), `https://tools.ietf.org/html/rfc7011/`, 2013.

[8] B. Claise: "Cisco systems NetFlow services export version 9", IETF, RFC 3954 (Informational), `https://tools.ietf.org/html/rfc3954/`, 2013.

[9] A. Sperotto, R. Sadre, P. T. de Boer, A. Pras: "Hidden Markov model modeling of SSH brute-force attacks". In: C. Bartolini, L. P. Gaspary (eds.) *Integrated Management of Systems, Services, Processes and People in IT (DSOM 2009*, LNCS, vol. 5841, pp. 164–176, Springer, 2009.

[10] J. Vykopal, T. Plesnik, P. Minarik: "Network-based dictionary attack detection".

In: *International Conference on Future Networks (ICFN 2009)*, pp. 23–27, IEEE, 2009.

[11] R. Hofstede, L. Hendriks, A. Sperotto, A. Pras: "SSH compromise detection using NetFlow/IPFIX". *ACM SIGCOMM Computer Communication Review*, vol. 44(5), pp. 20–26, ACM, 2014.

[12] S. Axelsson: "The base-rate fallacy and the difficulty of intrusion detection". *ACM Transactions on Information and System Security*, vol. 3(3), pp. 186–205, ACM, 2000.

[13] V. Chandola, A. Banerjee, V. Kumar: "Anomaly detection: A survey". *ACM Computing Surveys*, vol. 41(3), pp. 15:1–15:58, ACM, 2009.

[14] R. Sommer, V. Paxson: "Outside the closed world: On using machine learning for network intrusion detection". In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy (S&P'10)*, pp. 305–316, IEEE, 2010.

[15] K. Rieck: "Computer security and machine learning: Worst enemies or best friends?" In: *1st SysSec Workshop (SysSec 2011)*, pp. 107–110, IEEE, 2011.

[16] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, J. Srivastava: "A comparative study of anomaly detection schemes in network intrusion detection". In: D. Barbara, C. Kamath (eds.) *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM 2003)*, pp. 25–36, SIAM, 2003.

[17] A. Patcha, J.-M. Park: "An overview of anomaly detection techniques: Existing solutions and latest technological trends". *Computer Networks*, vol. 51(12), pp. 3448–3470, Elsevier, 2007.

[18] C. Kreibich, J. Crowcroft: "Honeycomb: Creating intrusion detection signatures using honeypots". *ACM SIGCOMM Computer Communication Review*, vol. 34(1), pp. 51–56, ACM, 2004.

[19] K. Wang, G. Cretu, S. J. Stolfo: "Anomalous payload-based worm detection and signature generation". In: A. Valdes, D. Zamboni (eds.) *Recent Advances in Intrusion Detection (RAID 2005)*, LNCS, vol. 3858, pp. 227–246, Springer, 2005.

[20] M. M. Z. E. Mohammed, H. A. Chan, N. Ventura: "Honeycyber: Automated signature generation for zero-day polymorphic worms". In: *2008 IEEE Military Communications Conference (MILCOM 2008)*, IEEE, 2008.

[21] D. Anderson, T. Frivold, A. Valdes: "Next-generation intrusion detection expert system (NIDES): A summary", SRI International (Menlo Park, USA), Technical Report, 1995.

[22] W. Lee, S. J. Stolfo: "A framework for constructing features and models for intrusion detection systems". *ACM Transactions on Information and System Security*, vol. 3(4), pp. 227–261, ACM, 2000.

[23] O. Depren, M. Topallar, E. Anarim, M. K. Ciliz: "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks". *Expert Systems with Applications: An International Journal*, vol. 29(4), pp. 713–722, Elsevier, 2005.

[24] K. Hwang, Y. Chen, H. Liu: "Defending distributed systems against malicious intrusions and network anomalies". In: *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2005)*, IEEE, 2005.

[25] J. Zhang, M. Zulkernine: "A hybrid network intrusion detection technique using random forests". In: *1st International Conference on Availability, Reliability and Security (ARES 2006)*, pp. 262–269, IEEE, 2006.

[26] P. García-Teodoro, J. E. Diaz-Verdejo, J. E. Tapiador, R. Salazar-Hernandez: "Automatic generation of HTTP intrusion signatures by selective identification of anomalies". *Computers & Security*, vol. 55(c), pp. 159–174, Elsevier, 2015.

[27] G. Creech, J. Hu: "Generation of a new IDS test dataset: Time to retire the KDD collection". In: *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 4487–4492, IEEE, 2013.

[28] S. Abt, H. Baier: "Are we missing labels? A study of the availability of ground-truth in network security research". In: *Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS 2014)*, pp. 40–55, IEEE, 2014.

[29] M. Małowidzki, P. Berezinski, M. Mazur: "Network intrusion detection: Half a kingdom for a good dataset". In: *NATO STO SAS-139 Workshop*, 2015.

[30] R. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, M. A. Zissman: "Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation". In: *Proceedings DARPA Information Survivability Conference and Exposition (DISCEX'00)*, vol. 2, pp. 12–26, IEEE, 2000.

[31] R. Lippmann, J. Haines, D. Fried, J. Korba, K. Das: "The 1999 DARPA off-line intrusion detection evaluation". *Computer Networks*, vol. 34(4), pp. 579–595, Elsevier, 2000.

[32] J. Haines, R. Lippmann, D. Fried, M. Zissman, E. Tran, S. Boswell: "1999 DARPA intrusion detection evaluation: Design and procedures", MIT Lincoln Laboratory (Lexington, USA), Technical Report, 2001.

[33] A. Sperotto, R. Sadre, F. van Vliet, A. Pras: "A labeled data set for flow-based intrusion detection". In: G. Nunzi, C. M. Scoglio, X. Li (eds.) *IP Operations and Management (IPOM'09)*, LNCS, vol. 5843, pp. 39–50, Springer, 2009.

[34] S. García, M. Grill, H. Stiborek, A. Zunino: "An empirical comparison of botnet detection methods". *Computers & Security*, vol. 45, pp. 100–123, Elsevier, 2014.

[35] J. J. Santanna, R. van Rijswijk-Deij, A. Sperotto, R. Hofstede, M. Wierbosch, L. Z. Granville, A. Pras: "Booters - an analysis of DDoS-as-a-service attacks". In: R. Badonnel, J. Xiao, S. Ata, F. D. Turck, V. Groza, C. R. P. dos Santos (eds.) *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 243–251, IEEE, 2015.

[36] F. Beer, T. Hofer, D. Karimi, U. Bühler: "A new attack composition for network security". In: P. Müller, B. Neumair, H. Reiser, G. Dreo Rodosek (eds.) *10. DFN-Forum Kommunikationstechnologie*, LNI, vol. P-271, pp. 11–20, GI, 2017.

[37] B. Trammell, E. Boschi: "An introduction to IP flow information export (IPFIX)". *IEEE Communications Magazine*, vol. 49(4), pp. 89–95, IEEE, 2011.

[38] B. Claise, B. Trammell: "Information model for IP flow information export (IPFIX)", IETF, RFC 7012 (Proposed Standard), `https://tools.ietf.org/html/rfc7012/`, 2013.

[39] G. Sadasivan, N. Brownlee, B. Claise, J. Quittek: “Architecture for IP flow information export”, IETF, RFC 5470 (Informational), https://tools.ietf.org/html/rfc5470/, 2009.

[40] B. Trammell, E. Boschi: “Bidirectional flow export using IP flow information export (IPFIX)”, IETF, RFC 5103 (Proposed Standard), https://tools.ietf.org/html/rfc5103/, 2008.

[41] A. Kobayashi, B. Claise, G. Muenz, K. Ishibashi: “IP flow information export (IPFIX) mediation: Framework”, IETF, RFC 6183 (Informational), https://tools.ietf.org/html/rfc6183/, 2011.

[42] J. P. Anderson: “Computer security threat monitoring and surveillance”, J. P. Anderson Company (Fort Washington, USA), Technical Report, 1980.

[43] D. E. Denning: “An intrusion-detection model”. *IEEE Transactions on Software Engineering*, vol. SE-13(2), pp. 222–232, IEEE, 1987.

[44] T. F. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, C. Jalali, P. G. Neumann, H. S. Javitz, A. Valdes, T. D. Garvey: “A real-time intrusion-detection expert system (IDES)”, SRI International (Menlo Park, USA), Technical Report, 1992.

[45] M. Sebring, E. Shellhouse, M. Hanna, R. Whitehurst: “Expert systems in intrusion detection: A case study”. In: *Proceedings of the 11th National Computer Security Conference*, pp. 74–81, 1988.

[46] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, D. Wolber: “A network security monitor”. In: *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 296–304, IEEE, 1990.

[47] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, D. Mansur: “DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype”. In: *Proceedings of the 14th National Computer Security Conference*, vol. 1, pp. 167–176, 1991.

[48] C. Gruhl, F. Beer, H. Heck, B. Sick, U. Bühler, A. Wacker, S. Tomforde: “A concept for intelligent collaborative network intrusion detection”. In: C. Trinitis, T. Pionteck (eds.) *ARCS 2017: 30th GI/ITG International Conference on Architecture of Computing Systems: Workshop Proceedings (SAOS 2017)*, pp. 78–85, VDE, 2017.

[49] U. Lindqvist, P. A. Porras: “Detecting computer and network misuse through the production-based expert system toolset (P-BEST)”. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pp. 146–161, IEEE, 1999.

[50] M. Roesch: “Snort: Lightweight intrusion detection for networks”. In: *Proceedings of the 13th USENIX Conference on System Administration (LISA’99)*, pp. 229–238, USENIX, 1999.

[51] S. Staniford, J. A. Hoagland, J. M. McAlerney: “Practical automated detection of stealthy portscans”. *Journal of Computer Security*, vol. 10, pp. 105–136, IOS Press, 2002.

[52] K. Wang, S. J. Stolfo: “Anomalous payload-based network intrusion detection”. In: E. Jonsson, A. Valdes, M. Almgren (eds.) *Recent Advances in Intrusion Detection*, LNCS, vol. 3224, pp. 203–222, Springer, 2004.

[53] T. M. Mitchell: "Machine learning". McGraw-Hill, 1997.

[54] F. Rosenblatt: "The perceptron - A perceiving and recognizing automaton", Cornell Aeronautical Laboratory, Inc. (Buffalo, USA), Technical Report, 1957.

[55] J. MacQueen: "Some methods for classification and analysis of multivariate observations". In: L. M. Le Cam, J. Neyman (eds.) *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, University of California Press, 1967.

[56] R. S. Michalski: "On the quasi-minimal solution of the covering problem". In: *Proceedings of the 5th International Symposium on Information Processing (FCIP-69)*, vol. A3, pp. 125–128, 1969.

[57] R. S. Michalski: "A theory and methodology of inductive learning". In: R. S. Michalski, J. G. Carbonell, T. M. Mitchell (eds.) *Machine Learning: An Artificial Intelligence Approach*, pp. 111–161, Tioga Publishing, 1983.

[58] J. R. Quinlan: "Induction of decision trees". *Machine Learning*, vol. 1, pp. 81–106, Kluwer, 1986.

[59] J. R. Quinlan: "C4.5: programs for machine learning". Morgan Kaufmann, 1993.

[60] T. Kohonen: "The self-organizing map". *Proceedings of the IEEE*, vol. 78(9), pp. 1464–1480, IEEE, 1990.

[61] J. Cendrowska: "PRISM: An algorithm for inducing modular rules". *International Journal of Man-Machine Studies*, vol. 27(4), pp. 349–370, 1987.

[62] C. Cortes, V. Vapnik: "Support-vector network". *Machine Learning*, vol. 20, pp. 273–297, Kluwer, 1995.

[63] C. J. C. Burges: "A tutorial on support vector machines for pattern recognition". *Data Mining and Knowledge Discovery*, vol. 2(2), pp. 121–167, Kluwer, 1998.

[64] P.-N. Tan, M. Steinbach, V. Kumar: "Introduction to data mining". Pearson, 2006.

[65] P. Flach: "Machine learning: The art and science of algorithms that make sense of data". Cambridge University Press, 2012.

[66] G. James, D. Witten, T. Hastie, R. Tibshirani: "An introduction to statistical learning". STS, Springer, 2013.

[67] T. Hastie, R. Tibshirani, J. Friedman: "The elements of statistical learning". SSS, 2nd edition, Springer, 2008.

[68] J. Gama: "Knowledge discovery from data streams". Chapman & Hall, 2010.

[69] A. K. Jain, R. C. Dubes: "Algorithms for clustering data". Prentice Hall, 1988.

[70] A. K. Jain, M. N. Murty, P. J. Flynn: "Data clustering: A review". *ACM Computing Surveys*, vol. 31(3), pp. 264–323, ACM, 1999.

[71] R. Xu, D. Wunsch: "Survey of clustering algorithms". *IEEE Transactions on Neural Networks*, vol. 16(3), pp. 645–678, IEEE, 2005.

[72] G. Hamerly, C. Elkan: "Alternatives to the k-means algorithm that find better clusterings". In: *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM'02)*, pp. 600–607, ACM, 2002.

[73] C. Fraley, A. E. Raftery: "How many clusters? which clustering method? answers

via model-based cluster analysis". *The Computer Journal*, vol. 41(8), pp. 578–588, Oxford University Press, 1998.

[74] D. Pelleg, A. W. Moore: "X-means: Extending k-means with efficient estimation of the number of clusters". In: *Proceedings of the 17th International Conference on Machine Learning (ICML'00)*, pp. 727–734, Morgan Kaufmann, 2000.

[75] C. A. Sugar, G. M. James: "Finding the number of clusters in a dataset: An information-theoretic approach". *Journal of the American Statistical Association*, vol. 98(463), pp. 750–763, Taylor & Francis, 2003.

[76] W.-C. Chen, R. Maitra, V. Melnykov, D. Nettleton, D. Faden, R. Rostamian: "EM algorithm for model-based clustering of finite mixture gaussian distribution (0.2-12)", `https://cran.r-project.org/web/packages/EMCluster/EMCluster.pdf`, 2019.

[77] M. H. Bhuyan, D. K. Bhattacharyya, J. K. Kalita: "Network anomaly detection: Methods, systems and tools". *IEEE Communications Surveys & Tutorials*, vol. 16(1), pp. 303–336, IEEE, 2014.

[78] A. A. Ghorbani, W. Lu, M. Tavallaee: "Network intrusion detection and prevention - Concepts and techniques". Advances in Information Security, Springer, 2010.

[79] O. Chapelle, B. Schölkopf, A. Zien (eds.): "Semi-supervised learning". MITP, 2006.

[80] X. Zhu: "Semi-supervised learning literature survey", University of Wisconsin-Madison (Madison, USA), Technical Report, 2008.

[81] D. Yarowsky: "Unsupervised word sense disambiguation rivaling supervised methods". In: *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics (ACL'95)*, pp. 189–196, ACM, 1995.

[82] A. Blum, T. Mitchell: "Combining labeled and unlabeled data with co-training". In: *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT'98)*, pp. 92–100, ACM, 1998.

[83] A. Demiriz, K. P. Bennett, M. J. Embrechts: "Semi-supervised clustering using genetic algorithms". In: *Proceedings of the 1999 Artificial Neural Networks in Engineering Conference (ANNIE'99)*, pp. 809–814, 1999.

[84] R. Dara, S. C. Kremer, D. A. Stacey: "Clustering unlabeled data with SOMs improves classification of labeled real-world data". In: *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN'02)*, vol. 3, pp. 2237–2242, 2002.

[85] K. Nigam, A. K. Mccallum, S. Thrun, T. Mitchell: "Text classification from labeled and unlabeled documents using EM". *Machine Learning*, vol. 39, pp. 103–134, 2000.

[86] B. Settles: "Active learning literature survey", University of Wisconsin-Madison (Madison, USA), Technical Report, 2010.

[87] C. C. Aggarwal, X. Kong, Q. Gu, J. Han, P. Yu: "Active learning: A survey". In: C. C. Aggarwal (ed.) *Data Classification: Algorithms and Applications*, pp. 572–605, CRC Press, 2014.

[88] D. Lewis, W. Gale: "A sequential algorithm for training text classifiers". In:

B. Croft, C. J. van Rijsbergen (eds.) *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, pp. 3–12, Springer, 1994.

[89] H. S. Seung, M. Opper, H. Sompolinsky: "Query by committee". In: *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT'92)*, pp. 287–294, ACM, 1992.

[90] K. Tomanek, F. Olsson: "A web survey on the use of active learning to support annotation of text data". In: *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing (ALNLP'09)*, pp. 45–48, ACL, 2009.

[91] A. Beygelzimer, D. Hsu, J. Langford, T. Zhang: "Agnostic active learning without constraints". In: J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, A. Culotta (eds.) *Proceedings of the 23rd International Conference on Neural Information Processing Systems (NIPS'10)*, vol. 1, pp. 199–207, Curran Associates Inc., 2010.

[92] P. Bachman, A. Sordoni, A. Trischler: "Learning algorithms for active learning". In: D. Precup, Y. W. Teh (eds.) *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, vol. 70, pp. 301–310, PMLR, 2017.

[93] T. Le, F. Stahl, M. M. Gaber, J. B. Gomes, G. Di Fatta: "On expressiveness and uncertainty awareness in rule-based classification for data streams". *Neurocomputing*, vol. 265(C), pp. 127–141, Elsevier, 2017.

[94] C. Molnar: "Interpretable machine learning", `https://christophm.github.io/interpretable-ml-book/`, 2019.

[95] M. T. Ribeiro, S. Singh, 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)*, pp. 1135–1144, ACM, 2016.

[96] C. Rudin: "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". *Nature Machine Intelligence*, vol. 1, pp. 206–215, Nature Publishing Group, 2019.

[97] J. R. Quinlan: "Simplifying decision trees". *International Journal of Man-Machine Studies*, vol. 27(3), pp. 221–234, 1987.

[98] J. Gama, P. Kosina: "Learning decision rules from data streams". In: T. Walsh (ed.) *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, vol. 2, pp. 1255–1260, AAAI Press, 2011.

[99] I. H. Witten, E. Frank, M. A. Hall: "Data mining: Practical machine learning tools and techniques". 3rd edition, Morgan Kaufmann, 2011.

[100] J. Fürnkranz, D. Gamberger, N. Lavrač: "Foundations of rule learning". Springer, 2012.

[101] B. Letham, C. Rudin, T. H. McCormick, D. Madigan: "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model". *The Annals of Applied Statistics*, vol. 9(3), pp. 1350–1371, 2015.

[102] G. Ditzler, M. Roveri, C. Alippi, R. Polikar: "Learning in nonstationary environments: A survey". *IEEE Computational Intelligence Magazine*, vol. 10(4), pp.

12–25, IEEE, 2015.

[103] T. Lane, C. E. Brodley: "Temporal sequence learning and data reduction for anomaly detection". *ACM Transactions on Information and System Security (TISSEC)*, vol. 2(3), pp. 295–331, ACM, 1999.

[104] A. Rozsypal, M. Kubat: "Association mining in time-varying domains". *Intelligent Data Analysis*, vol. 9(3), pp. 273–288, IOS Press, 2005.

[105] M. Kukar: "Drifting concepts as hidden factors in clinical studies". In: M. Dojat, E. T. Keravnou, P. Barahona (eds.) *Artificial Intelligence in Medicine*, LNCS, vol. 2780, pp. 355–364, Springer, 2003.

[106] I. Žliobaitė: "Learning under concept drift: An overview", Vilnius University (Vilnius, Lithuania), Technical Report, 2010.

[107] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia: "A survey on concept drift adaptation". *ACM Computing Surveys*, vol. 46(4), pp. 44:1–44:37, ACM, 2014.

[108] I. Žliobaitė, M. Pechenizkiy, J. Gama: "An overview of concept drift applications". In: N. Japkowicz, J. Stefanowski (eds.) *Big data analysis: New algorithms for a new society*, SBD, vol. 16, pp. 91–114, Springer, 2016.

[109] J. C. Schlimmer, R. H. Granger: "Incremental learning from noisy data". *Machine Learning*, vol. 1(3), pp. 317–354, Kluwer, 1986.

[110] J. C. Schlimmer, R. H. Granger: "Beyond incremental processing: Tracking concept drift". In: *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI'86)*, pp. 502–507, AAAI Press, 1986.

[111] M. M. Gaber, A. Zaslavsky, S. Krishnaswamy: "Mining data streams: A review". *ACM SIGMOD Record*, vol. 34(2), pp. 18–26, ACM, 2005.

[112] A. Bifet, R. Gavalda, G. Holmes, B. Pfahringer: "Machine learning for data streams with practical examples in MOA". MITP, 2018.

[113] R. O. Duda, P. E. Hart, D. G. Stork: "Pattern classification". 2nd edition, Wiley-Interscience, 2000.

[114] C. M. Bishop: "Pattern recognition and machine learning". Springer, 2007.

[115] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, F. Petitjean: "Characterizing concept drift". *Data Mining and Knowledge Discovery*, vol. 30(4), pp. 964–994, Springer, 2016.

[116] A. Tsymbal: "The problem of concept drift: Definitions and related work", Trinity College (Dublin, Ireland), Technical Report, 2004.

[117] J. Stefanowski: "Stream classification". In: C. Sammut, G. I. Webb (eds.) *Encyclopedia of Machine Learning and Data Mining*, pp. 1191–1199, 2nd edition, Springer, 2017.

[118] M. Salganicoff: "Tolerating concept and sampling shift in lazy learning using prediction error context switching". *Artificial Intelligence Review*, vol. 11(1), pp. 133–155, Kluwer, 1997.

[119] D. J. Hand: "Construction and assessment of classification rules". Wiley, 1997.

[120] J. Gao, W. Fan, J. Han, P. S. Yu: "A general framework for mining concept-drifting data streams with skewed distributions". In: C. Apte, B. Lui, S. Parthasarathy,

D. Skillicorn (eds.) *Proceedings of the 7th SIAM International Conference on Data Mining (SDM 2007)*, pp. 3–14, SIAM, 2007.

[121] M. M. Lazarescu, S. Venkatesh, H. H. Bui: "Using multiple windows to track concept drift". *Intelligent Data Analysis*, vol. 8(1), pp. 29–59, IOS Press, 2004.

[122] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, N. D. Lawrence: "Dataset shift in machine learning". MITP, 2009.

[123] S. J. Delany, P. Cunningham, A. Tsymbal, L. Coyle: "A case-based technique for tracking concept drift in spam filtering". *Knowledge-Based Systems*, vol. 18(4–5, pp. 187–195, Elsevier, 2005.

[124] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, F. Herrera: "A unifying view on dataset shift in classification". *Pattern Recognition*, vol. 45(1), pp. 521–530, Elsevier, 2012.

[125] "Big data", Gartner Inc. (IT Glossary), `https://gartner.com/en/information-technology/glossary/big-data/`, 2021.

[126] M. Schroeck, R. Shockley, J. Smart, D. Romero-Morales, P. Tufano: "Analytics: The real-world use of big data", IBM Institute for Business Value (Armonk, USA) and University of Oxford (Oxford, UK), White Paper, 2012.

[127] W. Fan, A. Bifet: "Mining big data: Current status, and forecast to the future". *ACM SIGKDD Explorations Newsletter*, vol. 14(2), pp. 1–5, ACM, 2013.

[128] D. Boyd, K. Crawford: "Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon". *Information, Communication & Society*, vol. 15(5), pp. 662–679, Taylor & Francis, 2012.

[129] N. Marz, J. Warren: "Big data - Principles and best practises of scalable realtime data systems". Manning, 2015.

[130] J. Kreps: "Questioning the lambda architecture", O'Reilly Radar, `https://oreilly.com/radar/questioning-the-lambda-architecture/`, 2014.

[131] D. J. DeWitt, M. Stonebraker: "MapReduce: A major step backwards", The Database Column, `https://dsf.berkeley.edu/cs286/papers/backwards-vertica2008.pdf`, 2008.

[132] M. van Steen, A. S. Tanenbaum: "Distributed systems". 3rd edition, `https://distributed-systems.net`, 2017.

[133] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, D. Warneke: "The stratosphere platform for big data analytics". *The VLDB Journal*, vol. 23, pp. 939–964, Springer, 2014.

[134] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas: "Apache Flink™: Stream and batch processing in a single engine". *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 38(4), pp. 28–38, IEEE, 2015.

[135] "Apache Flink documentation", Apache Flink, `https://ci.apache.org/projects/flink/flink-docs-stable/`, 2021.

[136] E. F. Codd: "Derivability, redundancy and consistency of relations stored in large data banks". *ACM SIGMOD Record*, vol. 38(1), pp. 17–36, ACM, 1969.

[137] E. F. Codd: "A relational model of data for large shared data banks". *Communi-*

cations of the ACM, vol. 13(6), pp. 377–387, ACM, 1970.

[138] M. Stonebraker: "The case for shared nothing". *Database Engineering*, vol. 9(1), pp. 4–9, 1986.

[139] J. M. Hellerstein, M. Stonebraker, J. Hamilton: "Architecture of a database system". *Foundations and Trends in Databases*, vol. 1(2), pp. 141–259, 2007.

[140] M. T. Özsu, P. Valduriez: "Principles of distributed database systems". 3rd edition, Springer, 2011.

[141] L. B. Sokolinsky: "Survey of architectures of parallel database systems". *Programming and Computer Software*, vol. 30(6), pp. 337–346, 2004.

[142] G. J. Klir, M. J. Wierman: "Uncertainty-based information - Elements of generalized information theory", STUDFUZZ, vol. 15. , 2nd edition, Springer, 1999.

[143] Z. Pawlak: "Rough sets". *International Journal of Computer & Information Science*, vol. 11(5), pp. 341–356, Kluwer, 1982.

[144] Z. Pawlak: "Rough sets: Theoretical aspects of reasoning about data". Kluwer, 1991.

[145] L. A. Zadeh: "Fuzzy sets". *Information and Control*, vol. 8(3), pp. 338–353, 1965.

[146] G. Shafer: "A mathematical theory of evidence". Princeton University Press, 1976.

[147] I. Düntsch, G. Gediga: "Uncertainty measures of rough set prediction". *Artificial Intelligence*, vol. 106(1), pp. 109–137, Elsevier, 1998.

[148] Z. Pawlak, A. Skowron: "Rough sets and conflict analysis". In: J. Lu, G. Zhang, D. Ruan (eds.) *E-Service Intelligence: Methodologies, Technologies and Applications*, SCI, vol. 37, pp. 35–74, Springer, 2007.

[149] K. J. Cios, W. Pedrycz, R. W. Swiniarski: "Rough sets". In: *Data Mining Methods for Knowledge Discovery*, SECS, vol. 458, pp. 27–71, Springer, 1998.

[150] A. Skowron, C. Rauszer: "The discernibility matrices and functions in information systems". In: R. Słowiński (ed.) *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*, TDLD, vol. 11, pp. 331–362, Springer, 1992.

[151] R. Jensen, Q. Shen: "Computational intelligence and feature selection: Rough and fuzzy approaches". IEEE Press Series on Computational Intelligence, Wiley-IEEE, 2008.

[152] A. Chouchoulas, Q. Shen: "Rough set-aided keyword reduction for text categorization". *Applied Artificial Intelligence*, vol. 15(9), pp. 843–873, Taylor & Francis, 2001.

[153] W. Ziarko: "Variable precision rough set model". *Journal of Computer and System Sciences*, vol. 46(1), pp. 39–59, 1993.

[154] J. Komorowski, Z. Pawlak, L. Polkowski, A. Skowron: "Rough sets: A tutorial". In: S. K. Pal, A. Skowron (eds.) *Rough Fuzzy Hybridization: A New Trend in Decision-Making*, pp. 3–98, Springer, 1999.

[155] G. Holmes, A. Donkin, I. H. Witten: "WEKA: A machine learning workbench". In: *Proceedings of the Australian New Zealnd Intelligent Information Systems Conference (ANZIIS'94)*, pp. 357–361, 1994.

[156] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, B. Wiswedel: "KNIME: The konstanz information miner". In: C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (eds.) *Data Analysis, Machine Learning and Applications*, pp. 319–326, STUDIES CLASS, Springer, 2008.

[157] I. Mierswa, M. W. R. Klinkenberg, M. Scholz, T. Euler: "YALE: Rapid prototyping for complex data mining tasks". In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pp. 935–940, ACM, 2006.

[158] A. Ohrn, J. Komorowski: "ROSETTA - A rough set toolkit for analysis of data". In: *Proceedings of the 3rd International Joint Conference on Information Sciences*, pp. 403–407, 1997.

[159] J. G. Bazan, M. Szczuka: "The rough set exploration system". In: J. F. Peters, A. Skowron (eds.) *Transactions on Rough Sets III*, LNCS, vol. 3400, pp. 37–56, Springer, 2005.

[160] B. Prędki, R. Słowiński, J. Stefanowski, R. Susmaga, S. Wilk: "ROSE - Software implementation of the rough set theory". In: L. Polkowski, A. Skowron (eds.) *Rough Sets and Current Trends in Computing (RSCTC 1998)*, LNCS, vol. 1424, pp. 1–2, Springer, 1999.

[161] T. Tileston: "Have your cake & eat it too! Accelerate data mining combining SAS & Teradata". In: *Teradata Partners 2005 (Experience the Possibilities)*, 2005.

[162] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, C. Welton: "MAD Skills: New analysis practices for big data". In: *Proceedings of the VLDB Endowment*, vol. 2(2), pp. 1481–1492, VLDB Endowment, 2009.

[163] X. Feng, A. Kumar, B. Recht, C. Ré: "Towards a unified architecture for in-rdbms analytics". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD'12)*, pp. 325–336, ACM, 2012.

[164] S. Prasad, A. Fard, V. Gupta, J. Martinez, J. LeFevre, V. Xu, M. Hsu, I. Roy: "Large-scale predictive analytics in Vertica: Fast data transfer, distributed model creation, and in-database prediction". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD'15)*, pp. 1657–1668, ACM, 2015.

[165] U. Syed, S. Vassilvitskii: "SQML: Large-scale in-database machine learning with pure SQL". In: *Proceedings of the 2017 Symposium on Cloud Computing (SoCC'17)*, p. 659, ACM, 2017.

[166] X. Hu: "Knowledge discovery in databases: An attribute-oriented rough set approach", University of Regina (Regina, Canada), PhD Thesis, 1995.

[167] Y. Cai: "Attribute-oriented induction in relational databases", Simon Fraser University (Burnaby, Canada), Master Thesis, 1989.

[168] F. F. Machuca, M. Millán: "Enhancing the exploitation of data mining in relational database systems via the rough sets theory including precision variables". In: *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC'98)*, pp. 70–73, ACM, 1998.

[169] T. Y. Lin: "Rough set theory in very large databases". In: *Symposium on Modeling,*

Analysis and Simulation: IMACS Multiconference, Computational Engineering in Systems Applications (CESA'96), vol. 2, pp. 936–941, 1996.

[170] F. F. Machuca, M. Millán: "Enhancing query processing in extented relational database systems via rough set theory to exploit data mining potential". In: O. Pons, M. A. Vila, J. Kacprzyk (eds.) *Knowledge Management in Fuzzy Databases*, STUDFUZZ, vol. 39, pp. 349–370, Physica, 2000.

[171] M. C. Fernandez-Baizán, E. Menasalvas Ruiz, J. M. Peña Sánchez: "Integrating RDMS and data mining capabilities using rough sets". In: *Proceedings of the 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 1996)*, pp. 1439–1445, 1996.

[172] H. S. Nguyen: "Approximate boolean reasoning: Foundations and applications in data mining". In: J. F. Peters, A. Skowron (eds.) *Transactions on Rough Sets V*, LNCS, vol. 4100, pp. 114–122, Springer, 2006.

[173] X. Hu, T. Y. Lin, J. Han: "A new rough set model based on database systems". In: G. Wang, Q. Liu, Y. Yao, A. Skowron (eds.) *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC 2003)*, LNCS, vol. 2639, pp. 114–121, Springer, 2003.

[174] J. Han, X. Hu, T. Y. Lin: "A new computation model for rough set theory based on database systems". In: Y. Kambayashi, M. Mohania, W. Wöß(eds.) *Data Warehousing and Knowledge Discovery (DaWak 2003)*, LNCS, vol. 2737, pp. 381–390, Springer, 2003.

[175] A. Kumar: "New techniques for data reduction in a database system for knowledge discovery applications". *Journal of Intelligent Information Systems*, vol. 10, pp. 31–48, Kluwer, 1998.

[176] K. Vaithyanathan, T. Y. Lin: "High frequency rough set model based on database systems". In: *Proceedings of the 2008 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS 2008)*, pp. 1–6, 2008.

[177] K. Czajkowski, M. Drabowski: "Semantic data selections and mining in decision tables". In: T. Czachórski, S. Kozielski, U. Stańczyk (eds.) *Man-Machine Interactions 2*, AINSC, vol. 103, pp. 279–286, Springer, 2011.

[178] H. Sun, Z. Xiong, Y.Wang: "Research on integrating ordbms and rough set theory". In: S. Tsumoto, R. Słowiński, J. Komorowski, J. W. Grzymała-Busse (eds.) *Rough Sets and Current Trends in Computing (RSCTC 2004)*, LNCS, vol. 3066, pp. 169–175, Springer, 2004.

[179] C.-C. Chan: "Learning rules from very large databases using rough multisets". In: J. F. Peters, A. Skowron, J. W. Grzymała-Busse, B. Kostek, R. W. Świniarski, M. S. Szczuka (eds.) *Transactions on Rough Sets I*, LNCS, vol. 3100, pp. 59–77, Springer, 2004.

[180] U. Seelam, C.-C. Chan: "A study of data reduction using multiset decision tables". In: T. Y. Lin, X. Hu, J. Han, X. Shen, Z. Li (eds.) *2007 IEEE International Conference on Granular Computing (GrC 2007)*, pp. 362–367, IEEE, 2007.

[181] S. Naouali, R. Missaoui: "Flexible query answering in data cubes". In: A. M. Tjoa, J. Trujillo (eds.) *Data Warehousing and Knowledge Discovery (DaWaK 2005)"*, LNCS, vol. 3589, pp. 221–232, Springer, 2005.

[182] T. Beaubouef, F. E. Petry: "A rough set model for relational databases". In: W. P. Ziarko (ed.) *Rough Sets, Fuzzy Sets and Knowledge Discovery (RSKD'93)*, pp. 100–107, Springer, 1994.

[183] L.-L. Wei, W. Zhang: "A method for rough relational database transformed into relational database". In: *Proceedings of the 2009 IITA International Conference on Services Science, Management and Engineering (SSME'09)*, pp. 50–52, IEEE, 2009.

[184] D. Slezak, J. Wroblewski, V. Eastwood, P. Synak: "Brighthouse: An analytic data warehouse for ad-hoc queries". In: *Proceedings of the VLDB Endowment*, vol. 1(2), pp. 1337–1345, VLDB Endowment, 2008.

[185] F. Beer: "Objektrelationale Datenbanken und Rough Sets für die Analyse von Contextualized Attention Metadata", Bonn-Rhine-Sieg University of Applied Sciences (St. Augustin, Germany), Master Thesis, 2009.

[186] F. Beer, U. Bühler: "In-database feature selection using rough set theory". In: J. P. Carvalho, M. J. Lesot, U. Kaymak, S. Vieira, B. Bouchon-Meunier, R. R. Yager (eds.) *Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 2016)*, CCIS, vol. 611, pp. 393–407, Springer, 2016.

[187] F. Beer, U. Bühler: "An in-database rough set toolkit". In: R. Bergmann, S. Görg, G. Müller (eds.) *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB (LWA'15)*, pp. 146–157, CEUR-WS.org, 2015.

[188] Z. Pawlak: "Information systems - Theoretical foundations". *Information Systems*, vol. 6(3), pp. 205–218, Pergamon Press, 1981.

[189] T. Y. Lin: "An overview of rough set theory from the point of view of relational databases". *Bulletin of International Rough Set Society*, vol. 1(1), pp. 30–34, 1997.

[190] R. Ramakrishnan, J. Gehrke: "Database management systems". 3rd edition, McGraw-Hill, 2002.

[191] H. García-Molina, J. D. Ullman, J. Widom: "Database systems - The complete book". 2nd edition, Pearson, 2009.

[192] S. H. Nguyen, H. S. Nguyen: "Some efficient algorithms for rough set methods". In: *Proceedings of the 6th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 1996)*, pp. 1451–1457, 1996.

[193] J. Peña, S. Létourneau, F. Famili: "Application of rough sets algorithms to prediction of aircraft component failure". In: D. J. Hand, N. J. Kok, M. R. Berthold (eds.) *Advances in Intelligent Data Analysis (IDA 1999)*, LNCS, vol. 1642, pp. 473–484, Springer, 1999.

[194] M. Beynon: "An investigation of β-reduct selection within the variable precision rough sets model". In: *Rough Sets and Current Trends in Computing (RSCTC 2000)*, LNCS, vol. 2005, pp. 114–122, Springer, 2001.

[195] M. Beynon: "Reducts within the variable precision rough sets model: A further investigation". *European Journal of Operational Research*, vol. 134(3), pp. 592–605, Elsevier, 2001.

[196] J.-S. Mi, W.-Z. Wu, W.-X. Zhang: "Approaches to knowledge reduction based on variable precision rough set model". *Information Sciences*, vol. 159(3), pp. 255–

272, Elsevier, 2004.

[197] L. S. Riza, A. Janusz, C. Bergmeir, C. Cornelis, F. Herrera, D. Ślęzak, J. M. Benítez: "Implementing algorithms of rough set theory and fuzzy rough set theory in the R package 'RoughSets'". *Information Sciences*, vol. 287, pp. 68–89, Elsevier, 2014.

[198] P. Baldi, P. Sadowski, D. Whiteson: "Searching for exotic particles in high-energy physics with deep learning". *Nature Communications*, vol. 5, pp. 4308:1–4308:14, 2014.

[199] A. Reiss, D. Stricker: "Introducing a new benchmarked dataset for activity monitoring". In: *16th International Symposium on Wearable Computers (ISWC 2012)*, pp. 108–109, IEEE, 2012.

[200] J. A. Blackard, D. J. Dean: "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables". *Computers and Electronics in Agriculture*, vol. 24(3), pp. 131–151, Elsevier, 1999.

[201] M. Tavallaee, E. Bagheri, W. Lu, A. A. Ghorbani: "A detailed analysis of the KDD cup 99 data set". In: *IEEE Symposium on Computational Intelligence in Security and Defense Applications (CISDA'2009)*, pp. 53–58, IEEE, 2009.

[202] J. G. Bazan, R. Latkowski, M. Szczuka: "DIXER - Distributed executor for rough set exploration system". In: D. Ślęzak, J. Yao, J. F. Peters, W. Ziarko, X. Hu (eds.) *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC 2005)*, LNCS, vol. 3642, pp. 39–47, Springer, 2005.

[203] "Email threat report for january-june 2018", FireEye Inc., `https://fireeye.com/offers/rpt-email-threat-report.html`, 2018.

[204] B. Krebs: "Wipro data breach", KrebsOnSecurity, `https://krebsonsecurity.com/tag/wipro-data-breach/`, 2019.

[205] B. Krebs: "DDoS on Dyn impacts Twitter, Spotify, Reddit", KrebsOnSecurity, `https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/`, 2016.

[206] A. Sfakianakis, C. Douligeris, L. Marinos, M. Lourenço, O. Raghimi: "ENISA threat landscape report 2018", European Union Agency for Cybersecurity, `https://enisa.europa.eu/publications/enisa-threat-landscape-report-2018/`, 2019.

[207] "2019 data breach investigations report", Verizon Inc., `https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf`, 2019.

[208] A. v. d. Stock, B. Glas, N. Smithline, T. Gigler: "OWASP top 10 - 2017: The ten most critical web application security risks", OWASP, `https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf`, 2017.

[209] "Web application vulnerability report 2019", Acunetix Ltd., `https://acunetix.com/blog/articles/acunetix-web-application-vulnerability-report-2019/`, 2019.

[210] "Sophos 2020 threat report", Sophos Ltd., `https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/sophoslabs-uncut-2020-threat-rep`

ort.pdf, 2019.

[211] T. Lane, C. E. Brodley: "Approaches to online learning and concept drift for user identification in computer security". In: R. Agrawal, P. Stolorz (eds.) *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pp. 259–263, AAAI Press, 1998.

[212] K. Sequeira, M. Zaki: "ADMIT: Anomaly-based data mining for intrusions". In: *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pp. 386–395, ACM, 2002.

[213] M. A. Maloof, R. S. Michalski: "A method for partial-memory incremental learning and its application to computer intrusion detection". In: *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'95)*, pp. 392–397, IEEE, 1995.

[214] W. Lee, S. J. Stolfo, K. W. Mok: "Adaptive intrusion detection: A data mining approach". *Artificial Intelligence Review*, vol. 14(6), pp. 533–567, Springer, 2000.

[215] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, P. Hakimian: "Detecting P2P botnets through network behavior analysis and machine learning". In: *2011 9th Annual International Conference on Privacy, Security and Trust (PST 2011)*, pp. 174–180, IEEE, 2011.

[216] F. Jemili, M. Zaghdoud, M. B. Ahmed: "A framework for an adaptive intrusion detection system using bayesian network". In: G. Muresan, T. Altiok, B. Melamed, D. Zeng (eds.) *2007 IEEE Intelligence and Security Informatics (ISI 2007)*, pp. 66–70, IEEE, 2007.

[217] D. Zhao, I. Traore, A. Ghorbani, B. Sayed, S. Saad, W. Lu: "Peer to peer botnet detection based on flow intervals". In: D. Gritzalis, S. Furnell, M. Theoharidou (eds.) *Information Security and Privacy Research (SEC 2012)*, IFIPAICT, vol. 376, pp. 87–102, Springer, 2012.

[218] G. Vigna, W. Robertson, D. Balzarotti: "Testing network-based intrusion detection signatures using mutant exploits". In: *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, pp. 21–30, ACM, 2004.

[219] S. Pastrana, A. Orfila, A. Ribagorda: "A functional framework to evade network IDS". In: R. H. Sprague (ed.) *Proceedings of the 44th Annual Hawaii International Conference on System Sciences (HICSS'11)*, pp. 1–10, IEEE, 2011.

[220] T. Cheng, Y. Lin, Y. Lai, P. Lin: "Evasion techniques: Sneaking through your intrusion detection/prevention systems". *IEEE Communications Surveys & Tutorials*, vol. 14(4), pp. 1011–1020, IEEE, 2012.

[221] I. Corona, G. Giacinto, F. Roli: "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues". *Information Sciences*, vol. 239, pp. 201–225, Elsevier, 2013.

[222] H. Heck, O. Kieselmann, A. Wacker: "Evaluating connection resilience for self-organizing cyber-physical systems". In: *The 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2016)*, pp. 140–141, IEEE, 2016.

[223] C. V. Zhou, C. Leckie, S. Karunasekera: "A survey of coordinated attacks and

collaborative intrusion detection". *Computers & Security*, vol. 29, pp. 124–140, Elsevier, 2010.

[224] E. Vasilomanolakis, S. Karuppayah, M. Mühlhäuser, M. Fischer: "Taxonomy and survey of collaborative intrusion detection". *ACM Computing Surveys*, vol. 47(4), pp. 55:1–55:33, ACM, 2015.

[225] R. Sommer, V. Paxson: "Enhancing byte-level network intrusion detection signatures with context". In: *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, pp. 262–271, ACM, 2003.

[226] V. Paxson: "Bro: A system for detecting network intruders in real-time". *Computer Networks*, vol. 31(23), pp. 2435–2463, 1999.

[227] D. Bolzoni, S. Etalle, P. Hartel: "POSEIDON: A 2-tier anomaly-based network intrusion detection system". In: *4th IEEE International Workshop on Information Assurance (IWIA'06)*, pp. 156–165, IEEE, 2006.

[228] M. Pihelgas: "A comparative analysis of open-source intrusion detection systems", Tallinn University of Technology (Tallinn, Estonia), Master Thesis, 2012.

[229] P.-C. Lin, J.-H. Lee: "Re-examining the performance bottleneck in a NIDS with detailed profiling". *Journal of Network and Computer Applications*, vol. 36(2), pp. 768–780, Elsevier, 2013.

[230] G. M. Jones, J. Stogoski: "Alternatives to signatures (alts)", CERT/CC (Pittsburgh, USA), White Paper 2014.

[231] M. Golling, R. Hofstede, R. Koch: "Towards multi-layered intrusion detection in high-speed networks". In: P. Brangetto, M. Maybaum, J. Stinissen (eds.) *2014 6th International Conference on Cyber Conflict (CyCon 2014)*, pp. 191–206, IEEE, 2014.

[232] "2016 global Internet phenomena - Spotlight: Encrypted Internet traffic", Sandvine Inc. (Waterloo, Canada), White Paper 2016.

[233] E. Ahlm, J. D'Hoinne, L. Orans, A. Hils: "Predicts 2017: Network and gateway security", Gartner Inc. (Stamford, USA), Technical Report, 2016.

[234] J. A. Lewis, D. E. Zheng, W. A. Carter: "The effect of encryption on lawful access to communications and data", Center for Strategic and International Studies (Washington, USA), Technical Report, 2017.

[235] R. Hackett: "The encryption wars are back on in congress. Here's what's at stake", Fortune, `https://fortune.com/2020/06/29/encryption-backdoors-computer-security-congress-heres-whats-at-stake/`, 2020.

[236] J. Steinberger, L. Schehlmann, S. Abt, H. Baier: "Anomaly detection and mitigation at Internet scale: A survey". In: G. Doyen, M. Waldburger, P. Čeleda, A. Sperotto, B. Stiller (eds.) *Emerging Management Mechanisms for the Future Internet*, LNCS, vol. 7943, pp. 49–60, Springer, 2013.

[237] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, B. Stiller: "An overview of IP flow-based intrusion detection". *IEEE Communications Surveys & Tutorials*, vol. 12(3), pp. 343–356, IEEE, 2010.

[238] M. F. Umer, M. Sher, Y. Bi: "Flow-based intrusion detection: Techniques and challenges". *Computers & Security*, vol. 70, pp. 238–254, Elsevier, 2017.

[239] L. Deri: "nProbe: An open source NetFlow probe for gigabit networks". In: *Proceedings of the TERENA Networking Conference (TNC 2003)*, pp. 1–4, 2003.

[240] S. Burschka, B. Dupasquier: "Tranalyzer: Versatile high performance network traffic analyser". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI 2016)*, pp. 1–8, IEEE, 2016.

[241] R. T. Lampert, C. Sommer, G. Münz, F. Dressler: "Vermont - A versatile monitoring toolkit for IPFIX and PSAMP". In: *Proceedings of IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*, pp. 62–65, 2006.

[242] C. M. Inacio, B. Trammell: "YAF: Yet another flowmeter". In: *Proceedings of the 24th Large Installation System Administration Conference (LISA'10)*, pp. 107–118, USENIX, 2010.

[243] M. Collins: "Network security through data analysis". O'Reilly, 2014.

[244] Y. Ding, M. Xiao, A.-W. Liu: "Research and implementation on Snort-based hybrid intrusion detection system". In: *Proceedings of the 8th International Conference on Machine Learning and Cybernetics*, pp. 1414–1418, IEEE, 2009.

[245] G. Kim, S. Lee, S. Kim: "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection". *Expert Systems with Applications*, vol. 41(4), pp. 1690–1700, Elsevier, 2014.

[246] C. Guo, Y. Ping, N. Liu, S.-S. Luo: "A two-level hybrid approach for intrusion detection". *Neurocomputing*, vol. 214, pp. 391–400, Elsevier, 2016.

[247] E. Nakashima, C. Timberg: "NSA officials worried about the day its potent hacking tool would get loose. Then it did.", Washington Post, `https://washingtonpost.com/business/technology/nsa-officials-worried-about-the-day-its-potent-hacking-tool-would-get-loose-then-it-did/2017/05/16/50670b16-3978-11e7-a058-ddbb23c75d82_story.html`, 2017.

[248] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru, A. Valdes: "Detecting unusual program behavior using the statistical component of the next-generation intrusion detection expert system (NIDES)", SRI International (Menlo Park, USA), Technical Report, 1995.

[249] P. A. Porras, P. G. Neumann: "EMERALD: Event monitoring enabling responses to anomalous live disturbances". In: *National Information Systems Security Conference*, pp. 361–370, 1997.

[250] O. Cepheli, S. Büyükçorak, G. K. Kurt: "Hybrid intrusion detection system for DDoS attacks". *Journal of Electrical and Computer Engineering*, pp. 1075648:1–1075648:8, Hindawi, 2016.

[251] U. Aslam, E. Batool, S. N. Ahsan, A. Sultan: "Hybrid network intrusion detection system using machine learning classification and rule based learning system". *International Journal of Grid and Distributed Computing*, vol. 10(2), 2017.

[252] Z. D. M. Seifeddine: "Hybrid intrusion detection system", University of South Australia (Adelaide, Australia), PhD Thesis, 2017.

[253] K. Hwang, M. Cai, Y. Chen, M. Qin: "Hybrid intrusion detection with weighted signature generation over anomalous Internet episodes". *IEEE Transactions on Dependable and Secure Computing*, pp. 41–55, IEEE, 2007.

[254] J. Zhang, M. Zulkernine, A. Haque: "Random-forests-based network intrusion detection systems". *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 38(5), pp. 649–659, IEEE, 2008.

[255] P. García-Teodoro, P. M. Muñoz-Feldstedt, D. Ruete-Zúñiga: "Automatic signature generation for network services through selective extraction of anomalous contents". In: T. Atmaca, J. Palicot, A. Nafkha, T. Tsiatsos, M. Marot, O. Dini (eds.) *The 6th Advanced International Conference on Telecommunications (AICT 2010)*, pp. 370–375, IEEE, 2010.

[256] M. E. Locasto, K. Wang, A. D. Keromytis, S. J. Stolfo: "FLIPS: Hybrid adaptive intrusion prevention". In: A. Valdes, D. Zamboni (eds.) *Recent Advances in Intrusion Detection (RAID 2005)*, LNCS, vol. 3858, pp. 82–101, Springer, 2005.

[257] K. Shafi, H. A. Abbass, W. Zhu: "An adaptive rule-based intrusion detection architecture". In: *Proceedings of the 2006 RNSA Security Technology Conference*, pp. 307–319, 2006.

[258] E. Tombini, H. Debar, L. Me, M. Ducasse: "A serial combination of anomaly and misuse IDSes applied to HTTP traffic". In: *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pp. 428–437, IEEE, 2004.

[259] D. Brumley, J. Newsome, D. Song, H. Wang, S. Jha: "Towards automatic generation of vulnerability-based signatures". In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P 2006)*, pp. 2–16, IEEE, 2006.

[260] G. Tahan, C. Glezer, Y. Elovici, L. Rokach: "Auto-Sign: An automatic signature generator for high-speed malware filtering devices". *Journal in Computer Virology*, vol. 6(2), pp. 91–103, Springer, 2009.

[261] K. Griffin, S. Schneider, X. Hu, T. Chiueh: "Automatic generation of string signatures for malware detection". In: E. Kirda, S. Jha, D. Balzarotti (eds.) *Recent Advances in Intrusion Detection*, LNCS, vol. 5758, pp. 101–120, Springer, 2009.

[262] H. Kim, B. Karp: "Autograph: Toward automated, distributed worm signature detection". In: *Proceedings of the 13th Conference on USENIX Security Symposium (SSYM'04)*, vol. 13, USENIX, 2004.

[263] H. Tu, Z. Li, B. Liu: "Mining network traffic for worm signature extraction". In: J. Ma, Y. Yin, J. Yu, S. Zhou (eds.) *5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2008)*, pp. 327–331, IEEE, 2008.

[264] E. Vasilomanolakis, S. Srinivasa, C. G. Cordero, M. Mühlhäuser: "Multi-stage attack detection and signature generation with ICS honeypots". In: S. O. Badonnel, M. Ulema, C. Cavdar, L. Z. Granville, C. R. P. dos Santos (eds.) *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, pp. 1227–1232, IEEE, 2016.

[265] G. Schaffrath, B.Stiller: "Conceptual integration of flow-based and packet-based network intrusion detection". In: D. Hausheer, J. Schönwälder (eds.) *Resilient Networks and Services (AIMS 2008)*, LNCS, vol. 5127, pp. 190–194, Springer, 2008.

[266] N. Duffield, P. Haffner, B. Krishnamurthy, H. Ringberg: "Rule-based anomaly detection on IP flows". In: *Proceedings of the 28th IEEE Conference on Computer Communications (INFOCOM 2009)*, pp. 424–432, IEEE, 2009.

[267] F. Erlacher, F. Dressler: "FIXIDS: A high-speed signature-based flow intrusion detection system". In: *Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, IEEE, 2018.

[268] N. Fallahi, A. Sami, M. Tajbakhsh: "Automated flow-based rule generation for network intrusion detection systems". In: *2016 24th Iranian Conference on Electrical Engineering (ICEE 2016)*, pp. 1948–1953, IEEE, 2016.

[269] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, W.-Y. Lin: "Intrusion detection by machine learning: A review". *Expert Systems with Applications*, vol. 36(10), pp. 11994–12000, Elsevier, 2009.

[270] A. L. Buczak, E. Guven: "A survey of data mining and machine learning methods for cyber security intrusion detection". *IEEE Communications Surveys & Tutorials*, vol. 18(2), pp. 1153–1176, IEEE, 2015.

[271] Z. B. Celik, J. Raghuram, G. Kesidis, D. J. Miller: "Salting public traces with attack traffic to test flow classifiers". In: *Proceedings of the 4th Conference on Cyber Security Experimentation and Test (CSET'11)*, USENIX, 2011.

[272] A. J. Avid, A. Haeberlen: "Challenges in experimenting with botnet detection systems". In: *Proceedings of the 4th Conference on Cyber Security Experimentation and Test (CSET'11)*, 2011.

[273] I. Perona, I. Gurrutxaga, O. Arbelaitz, J. I. Martin, J. Muguerza, J. M. Pérez: "Service-independent payload analysis to improve intrusion detection in network traffic". In: *Proceedings of the 7th Australasian Data Mining Conference (AusDM '08)*, pp. 171–178, Australian Computer Society, 2008.

[274] J. McHugh: "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory". *ACM Transactions on Information and System Security*, vol. 3(4), pp. 262–294, ACM, 2000.

[275] M. V. Mahoney, P. K. Chan: "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection". In: G. Vigna, C. Kruegel, E. Jonsson (eds.) *Recent Advances in Intrusion Detection (RAID 2003)*, LNCS, vol. 2820, pp. 220–237, Springer, 2003.

[276] A. Shiravi, H. Shiravi, M. Tavallaee, A. A. Ghorbani: "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". *Computers & Security*, vol. 31(3), pp. 357–374, Elsevier, 2012.

[277] W. Haider, J. Hu, J. Slay, B. Turnbull, Y. Xie: "Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling". *Journal of Network and Computer Applications*, vol. 87(1), pp. 185–192, Elsevier, 2017.

[278] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho: "Flow-based benchmark data sets for intrusion detection". In: M. Scanlon, N.-A. Le-Khac (eds.) *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS 2017)*, pp. 361–369, ACPI, 2017.

[279] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho: "Creation of flow-based data sets for intrusion detection". *Journal of Information Warfare*, vol. 16(4), pp. 40–53, 2017.

[280] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani: "Toward generating a new in-

trusion detection dataset and intrusion traffic characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, pp. 108–116, SciTePress, 2018.

[281] E. B. Beigi, H. H. Jazi, N. Stakhanova, A. A. Ghorbani: "Towards effective feature selection in machine learning-based botnet detection approaches". In: *Proceedings of the 2014 IEEE Conference on Communications and Network Security (CNS'14)*, pp. 247–255, IEEE, 2014.

[282] G. Szabó, D. Orincsay, S. Malomsoky, I. Szabó: "On the validation of traffic classification algorithms". In: M. Claypool, S. Uhlig (eds.) *Passive and Active Network Measurement (PAM 2008)*, LNCS, vol. 4979, pp. 72–81, 2008.

[283] B. Sangster, T. J. O'Connor, T. Cook, R. Fanelli, E. Dean, W. J. Adams, C. Morrell, G. Conti: "Toward instrumenting network warfare competitions to generate labeled datasets". In: *Proceedings of the 4th Conference on Cyber Security Experimentation and Test (CSET'09)*, 2009.

[284] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, A. Hotho: "A survey of network-based intrusion detection data sets". *Computers & Security*, vol. 86, pp. 147–167, Elsevier, 2019.

[285] R. Gerhards: "The syslog protocol", IETF, RFC 5424 (Proposed Standard), `https://tools.ietf.org/html/rfc5424/`, 2009.

[286] D. Brauckhoff, A. Wagner, M. May: "FLAME: A flow-level anomaly modeling engine". In: *Proceedings of the 2nd Workshop on Cyber Security Experimentation and Test (CSET'08)*, USENIX, 2008.

[287] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, M. Mühlhäuser: "ID2T: A DIY dataset creation toolkit for intrusion detection systems". In: *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS'15)*, pp. 739–740, IEEE, 2015.

[288] L. Ertöz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, P. Dokas: "The MINDS - Minnesota intrusion detection system". In: H. Kargupta, A. Joshi, K. Sivakumar, Y. Yesha (eds.) *Next Generation Data Mining*, pp. 199–218, MITP, 2004.

[289] A. Lakhina, M. Crovella, C. Diot: "Diagnosing network-wide traffic anomalies". In: *Proceedings of the 2004 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM'04)*, pp. 219–230, ACM, 2004.

[290] A. Lakhina, M. Crovella, C. Diot: "Mining anomalies using traffic feature distributions". In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM'05)*, pp. 217–228, ACM, 2005.

[291] K. Xu, Z. Zhang, S. Bhattacharyya: "Reducing unwanted traffic in a backbone network". In: *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*, USENIX, 2005.

[292] M. Rehák, M. Pěchouček, K. Bartoš, M. Grill, P. Celeda, V. Krmicek: "CAMNEP: An intrusion detection system for high speed networks". *Progress in Informatics*, vol. 5, pp. 65–74, 2008.

[293] J. François, S. Wang, R. State, T. Engel: "BotTrack: Tracking botnets using NetFlow and PageRank". In: J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, C. Scoglio (eds.) *NETWORKING 2011*, LNCS, vol. 6640, pp. 1–14, Springer, 2011.

[294] P. Narang, J. M. Reddy, C. Hota: "Feature selection for detection of peer-to-peer botnet traffic". In: *Proceedings of the 6th ACM India Computing Convention (Compute'13)*, pp. 16:1–16:9, ACM, 2013.

[295] F. Haddadi, A. N. Zincir-Heywood: "Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification". *IEEE Systems Journal*, vol. 10(4), pp. 1390–1401, IEEE, 2014.

[296] F. Iglesias, T. Zseby: "Analysis of network traffic features for anomaly detection". *Machine Learning*, vol. 101(1), pp. 59–84, Springer, 2015.

[297] L. Hellemons, L. Hendriks, R. Hofstede, R. S. A. Sperotto, A. Pras: "SSHCure: A flow-based SSH intrusion detection system". In: R. Sadre, J. Novotný, P. Čeleda, M. Waldburger, B. Stiller (eds.) *Dependable Networks and Services (AIMS 2012)*, LNCS, vol. 7279, pp. 86–97, Springer, 2012.

[298] M. Drašar: "Protocol-independent detection of dictionary attacks". *Advances in Communication Networking (EUNICE 2013)*, vol. 8115, pp. 304–309, Springer, 2013.

[299] J. Vykopal: "Flow-based brute-force attack detection in large and high-speed networks", Masaryk University (Brno, Czech Republic), PhD Thesis, 2013.

[300] V. Bukac, V. Matyas: "Analyzing traffic features of common standalone DoS attack tools". In: R. Chakraborty, P. Schwabe, J. Solworth (eds.) *Security, Privacy, and Applied Cryptography Engineering (SPACE 2015)*, LNCS, vol. 9354, pp. 21–40, Springer, 2015.

[301] D. Karimi: "IntErA host component - Entwicklung eines Frameworks zur Erfassung von Host-Features und verdächtigem Verhalten", University of Applied Sciences Fulda (Fulda, Germany), Bachelor Thesis, 2016.

[302] W. Lee, S. J. Stolfo, K. W. Mok: "A data mining framework for building intrusion detection models". In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (S&P 1999)*, pp. 120–132, IEEE, 1999.

[303] U. M. Fayyad, K. B. Irani: "Multi-interval discretization of continuous-valued attributes for classification learning". In: R. Bajcsy (ed.) *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, vol. 2, pp. 1022–1027, Morgan Kaufmann, 1993.

[304] K. Kenji, L. A. Rendell: "The feature selection problem: Traditional methods and a new algorithm". In: *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92)*, pp. 129–134, AAAI Press, 1992.

[305] M. A. Hall: "Correlation-based feature selection for machine learning", University of Waikato (Hamilton, New Zealand), PhD Thesis, 1999.

[306] D. Koller, M. Sahami: "Toward optimal feature selection". In: *Proceedings of the 13th International Conference on International Conference on Machine Learning (ICML'96)*, pp. 284–292, Morgan Kaufmann, 1996.

[307] R. Kohavi, G. H. John: "Wrappers for feature subset selection". *Artificial Intelligence*, vol. 97(1), pp. 273–324, Elsevier, 1997.

[308] I. Guyon, J. Weston, S. Barnhill, V. Vapnik: "Gene selection for cancer classification using support vector machines". *Machine Learning*, vol. 46, pp. 389–422, Kluwer, 2002.

[309] J. G. Dy, C. E. Brodley: "Feature subset selection and order identification for unsupervised learning". In: *Proceedings of the 17th International Conference on Machine Learning (ICML'00)*, pp. 247–254, Morgan Kaufmann, 2000.

[310] R. Tibshirani: "Regression shrinkage and selection via the lasso". *Journal of the Royal Statistical Society*, vol. 58(1), pp. 267–288, Wiley, 1996.

[311] "The R project for statistical computing", R Foundation for Statistical Computing, `https://r-project.org`, 2021.

[312] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, T. Hunt: "caret: Classification and regression training", `https://cran.r-project.org/web/packages/caret/caret.pdf`, 2020.

[313] K. Hornik, C. Buchta, A. Zeileis: "Open-source machine learning: R meets Weka". *Computational Statistics*, vol. 24(2), pp. 225–232, 2009.

[314] T. Hothorn, K. Hornik, A. Zeileis: "Unbiased recursive partitioning: A conditional inference framework". *Journal of Computational and Graphical Statistics*, vol. 15(3), pp. 651–674, 2006.

[315] I. H. W. E. Frank: "Generating accurate rule sets without global optimization". In: *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pp. 144–151, Morgan Kaufmann, 1998.

[316] W. W. Cohen: "Fast effective rule induction". In: A. Prieditis, S. Russell (eds.) *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, pp. 115–123, Morgan Kaufmann, 1995.

[317] L. Noriega: "Multilayer perceptron tutorial", Staffordshire University (Stafford, UK), Technical Report, 2005.

[318] D. S. Broomhead, D. Lowe: "Multivariable functional interpolation and adaptive networks". *Complex Systems*, vol. 2, pp. 321–355, 1988.

[319] C. Bouveyron, S. Girard, C. Schmid: "High dimensional discriminant analysis". *Communications in Statistics - Theory and Methods*, vol. 36(14), pp. 2607–2623, Taylor & Francis, 2007.

[320] D. K. Bhattacharyya, J. K. Kalita: "Network anomaly detection: A machine learning perspective". CRC Press, 2012.

[321] S. J. Russell, P. Norvig: "Artificial intelligence - A modern approach". 3rd edition, Prentice Hall, 2010.

[322] T. Fawcett: "An introduction to ROC analysis". *Pattern Recognition Letters*, vol. 27(8), pp. 861–874, Elsevier, 2006.

[323] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, J. Zhang: "Real time data mining-based intrusion detection". In: *Proceedings DARPA Information Survivability Conference and Exposition II (DISCEX'01)*, vol. 1, pp. 89–100, IEEE, 2001.

[324] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, E. Vázquez: "Anomaly-

based network intrusion detection: Techniques, systems and challenges". *Computers & Security*, vol. 28(1-2), pp. 18–28, Elsevier, 2009.

[325] D. Hawkins: "Identification of outliers". Chapman & Hall, 1980.

[326] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander: "LOF: Identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*, pp. 93–104, ACM, 2000.

[327] P. Laskov, P. Düssel, C. Schäfer, K. Rieck: "Learning intrusion detection: Supervised or unsupervised?" In: F. Roli, S. Vitulano (eds.) *Image Analysis and Processing - ICIAP 2005*, LNCS, vol. 3617, pp. 50–57, Springer, 2005.

[328] N. Görnitz, M. Kloft, K. Rieck, U. Brefeld: "Toward supervised anomaly detection". *Journal of Artificial Intelligence Research*, vol. 46(1), pp. 235–262, 2013.

[329] K. Hempstalk, F. Eibe: "Discriminating against new classes: One-class versus multi-class classification". In: W. Wobcke, M. Zhang (eds.) *AI 2008: Advances in Artificial Intelligence*, LNCS, vol. 5360, pp. 325–336, Springer, 2008.

[330] C. Bellinger, S. Sharma, N. Japkowicz: "One-class versus binary classification: Which and when?" In: *2012 11th International Conference on Machine Learning and Applications (ICMLA)*, vol. 2, pp. 102–106, IEEE, 2012.

[331] C. Bellinger, S. Sharma, O. R. Zaïane, N. Japkowicz: "Sampling a longer life: Binary versus one-class classification revisited". In: L. Torgo, B. Krawczyk, P. Branco, N. Moniz (eds.) *Proceedings of the 1st International Workshop on Learning with Imbalanced Domains: Theory and Applications*, pp. 64–78, 2017.

[332] C. Woolam, M. M. Masud, L. Khan: "Lacking labels in the stream: Classifying evolving stream data with few labels". In: J. Rauch, Z. W. Raś, P. Berka, T. Elomaa (eds.) *Foundations of Intelligent Systems (ISMIS 2009)*, LNCS, vol. 5722, pp. 552–562, Springer, 2009.

[333] I. Žliobaitė: "Change with delayed labeling: When is it detectable?" In: W. Fan, W. Hsu, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, X. Wu (eds.) *The 10th IEEE International Conference on Data Mining Workshops (ICDMW 2010)*, pp. 843–850, IEEE, 2010.

[334] M. M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. W. Hamlen, N. C. Oza: "Facing the reality of data stream classification: Coping with scarcity of labeled data". *Knowledge and Information Systems*, vol. 33(1), pp. 213–244, Springer, 2012.

[335] M. M. Masud, J. Gao, L. Khan, J. Han, B. Thuraisingham: "A practical approach to classify evolving data streams: Training with limited amount of labeled data". In: F. Giannotti, D. Gunopulos, F. Turini, C. Zaniolo, N. Ramakrishnan, X. Wu (eds.) *8th IEEE International Conference on Data Mining (ICDM 2008)*, pp. 929–934, IEEE, 2008.

[336] P. Zhang, X. Zhu, J. Tan, L. Guo: "Classifier and cluster ensembles for mining concept drifting data streams". In: G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, X. Wu (eds.) *2010 IEEE International Conference on Data Mining (ICDM 2010)*, pp. 1175–1180, IEEE, 2010.

[337] W. Fan, Y. Huang, H. Wang, P. S. Yu: "Active mining of data streams". In: M. W. Berry, U. Dayal, C. Kamath, D. Skillicorn (eds.) *Proceedings of the 4th SIAM*

International Conference on Data Mining (SDM 2004), pp. 457–461, SIAM, 2004.

[338] M. R. Kmieciak, J. Stefanowski: "Semi-supervised approach to handle sudden concept drift in Enron data". *Control and Cybernetics*, vol. 40(3), pp. 667–695, 2011.

[339] I. Žliobaitė, A. Bifet, B. Pfahringer, G. Holmes: "Active learning with drifting streaming data". *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25(1), pp. 27–39, IEEE, 2014.

[340] L. I. Kuncheva, J. S. Sánchez: "Nearest neighbour classifiers for streaming data with delayed labelling". In: F. Giannotti, D. Gunopulos, F. Turini, C. Zaniolo, N. Ramakrishnan, X. Wu (eds.) *8th IEEE International Conference on Data Mining (ICDM 2008)*, pp. 869–874, IEEE, 2008.

[341] G. R. Marrs, R. J. Hickey, M. M. Black: "The impact of latency on online classification learning with concept drift". In: Y. Bi, M. A. Williams (eds.) *Knowledge Science, Engineering and Management (KSEM 2010)*, LNCS, vol. 6291, pp. 459–469, Springer, 2010.

[342] K. B. Dyer, R. Capo, R. Polikar: "COMPOSE: A semisupervised learning framework for initially labeled nonstationary streaming data". *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25(1), pp. 12–26, IEEE, 2014.

[343] K. B. Dyer, R. Polikar: "Semi-supervised learning in initially labeled nonstationary environments with gradual drift". In: *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN'12)*, pp. 2741–2748, IEEE, 2012.

[344] V. M. A. Souza, D. F. Silva, J. Gama, G. E. A. P. A. Batista: "Data stream classification guided by clustering on nonstationary environments and extreme verification latency". In: S. Venkatasubramanian, J. Ye (eds.) *Proceedings of the 2015 SIAM International Conference on Data Mining (SDM 2015)*, pp. 873–881, SIAM, 2015.

[345] M. Umer, C. Frederickson, R. Polikar: "Learning under extreme verification latency quickly: FAST COMPOSE". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI 2016)*, pp. 1–8, IEEE, 2016.

[346] T. G. Dietterich: "Ensemble methods in machine learning". In: J. Kittler, F. Roli (eds.) *Multiple Classifier Systems (MCS'00)*, LNCS, vol. 1857, pp. 1–15, Springer, 2000.

[347] T. G. Dietterich: "Ensemble learning". In: M. A. Arbib (ed.) *The Handbook of Brain Theory and Neural Networks*, pp. 405–408, 2nd edition, MITP, 2002.

[348] L. I. Kuncheva: "Combining pattern classifiers: Methods and algorithms". Wiley, 2004.

[349] R. Polikar: "Ensemble based systems in decision making". *IEEE Circuits and Systems Magazine*, vol. 6(3), pp. 21–45, IEEE, 2006.

[350] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton: "Adaptive mixtures of local experts". *Neural Computation*, vol. 3(1), pp. 79–87, MITP, 1991.

[351] L. Xu, M. I. Jordan, G. E. Hinton: "An alternative model for mixtures of experts". In: *Advances in Neural Information Processing Systems 7 (NIPS)*, pp. 633–640, MITP, 1995.

[352] L. Xu, A. Krzyżak, C. Y. Suen: "Methods of combining multiple classifiers and their applications to handwriting recognition". *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22(3), pp. 418–43, IEEE, 1992.

[353] K. Woods, W. P. J. Kegelmeyer, K. Bowyer: "Combination of multiple classifiers using local accuracy estimates". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19(4), pp. 405–410, IEEE, 1997.

[354] Z.-H. Zhou: "Ensemble learning". In: S. Z. Li, A. Jain (eds.) *Encyclopedia of Biometrics*, pp. 270–273, Springer, 2009.

[355] L. Shapley, B. Grofman: "Optimizing group judgmental accuracy in the presence of interdependencies". *Public Choice*, vol. 43(3), pp. 329–343, Kluwer, 1984.

[356] L. Lam, C. Y. Suen: "Application of majority voting to pattern recognition: An analysis of its behavior and performance". *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27(5), pp. 553–568, IEEE, 1997.

[357] Z.-H. Zhou: "Ensemble methods: Foundations and algorithms". CRC Press, 2012.

[358] B. Efron, R. J. Tibshirani: "An introduction to the bootstrap". Springer, 1993.

[359] T. K. Ho: "The random subspace method for constructing decision forests". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20(8), pp. 832–844, IEEE, 1998.

[360] T. G. Dietterich, G. Bakiri: "Solving multiclass learning problems via error-correcting output codes". *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.

[361] L. Breiman: "Randomizing outputs to increase prediction accuracy". *Machine Learning*, vol. 40(3), pp. 229–242, Kluwer, 2000.

[362] L. Breiman: "Bagging predictors". *Machine Learning*, vol. 24(2), pp. 123–140, Kluwer, 1996.

[363] Y. Freund, R. E. Schapire: "Experiments with a new boosting algorithm". In: *Proceedings of the Thirteenth International Conference on Machine Learning (ICML'96)*, pp. 148–156, Morgan Kaufmann, 1996.

[364] Y. Freund, R. E. Schapire: "A decision-theoretic generalization of on-line learning and an application to boosting". *Journal of Computer and System Sciences*, vol. 55(1), pp. 119–139, Academic Press, 1997.

[365] R. Elwell, R. Polikar: "Incremental learning of concept drift in nonstationary environments". *IEEE Transactions on Neural Networks*, vol. 22(10), pp. 1517–1531, IEEE, 2011.

[366] M. Woźniak: "Application of combined classifiers to data stream classification". In: K. Saeed, R. Chaki, A. Cortesi, S. Wierzchoń (eds.) *Computer Information Systems and Industrial Management (CISIM 2013)*, LNCS, vol. 8104, pp. 13–23, Springer, 2013.

[367] J. Gama, P. Medas, G. Castillo, P. Rodrigues: "Learning with drift detection". In: A. L. C. Bazzan, S. Labidi (eds.) *Advances in Artificial Intelligence - SBIA 2004*, LNCS, vol. 3171, pp. 286–295, Springer, 2004.

[368] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, R. Morales-Bueno: "Early drift detection method". In: *Proceedings of the Inter-*

national Workshop on Knowledge Discovery from Data Streams (IWKDDS-2006), pp. 77–86, 2006.

[369] J. Gama, R. Sebastião, P. Pereira Rodrigues: "On evaluating stream learning algorithms". *Machine Learning*, vol. 90(3), pp. 317–346, Springer, 2013.

[370] L. I. Kuncheva: "Classifier ensembles for changing environments". In: F. Roli, J. Kittler, T. Windeatt (eds.) *Multiple Classifier Systems (MCS'04)*, LNCS, vol. 3077, pp. 1–15, Springer, 2004.

[371] N. Littlestone: "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm". *Machine Learning*, vol. 2, pp. 285–318, Kluwer, 1988.

[372] N. Littlestone, M. K. Warmuth: "The weighted majority algorithm". *Information and Computation*, vol. 108, pp. 212–261, 1994.

[373] A. Blum: "Empirical support for winnow and weighted-majority based algorithms: Results on a calendar scheduling domain". In: A. Prieditis, S. Russell (eds.) *Proceedings of the 12th International Conference on Machine Learning (ICML'95)*, pp. 64–72, Morgan Kaufmann, 1995.

[374] N. Oza: "Online ensemble learning", University of California (Berkeley, USA), PhD Thesis, 2001.

[375] A. Fern, R. Givan: "Online ensemble learning: An empirical study". *Machine Learning*, vol. 53(1–2), pp. 71–109, Kluwer, 2003.

[376] W. N. Street, Y. Kim: "A streaming ensemble algorithm SEA for large-scale classification". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pp. 377–382, ACM, 2001.

[377] H. Wang, W. Fan, P. S. Yu, J. Han: "Mining concept-drifting data streams using ensemble classifiers". In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pp. 226–235, ACM, 2003.

[378] J. Z. Kolter, M. A. Maloof: "Dynamic weighted majority: An ensemble method for drifting concepts". *Journal of Machine Learning Research*, vol. 8, pp. 2755–2790, 2007.

[379] J. Z. Kolter, M. A. Maloof: "Dynamic weighted majority: A new ensemble method for tracking concept drift". In: X. Wu, A. Tuzhilin, J. Shavlik (eds.) *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pp. 123–130, IEEE, 2003.

[380] D. Brzezinski, J. Stefanowski: "Reacting to different types of concept drift: The accuracy updated ensemble algorithm". *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25(1), pp. 81–94, IEEE, 2014.

[381] R. Polikar, L. Upda, S. S. Upda, V. Honavar: "Learn++: An incremental learning algorithm for supervised neural networks". *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, vol. 31(4), pp. 497–508, IEEE, 2001.

[382] G. Ditzler, R. Polikar: "Incremental learning of concept drift from streaming imbalanced data". *IEEE Transactions on Knowledge and Data Engineering*, vol. 25(10), pp. 2283–2301, IEEE, 2013.

[383] D. Brzezinski, J. Stefanowski: "Accuracy updated ensemble for data streams with concept drift". In: E. Corchado, M. Kurzyński, M. Woźniak (eds.) *Hybrid Artificial Intelligent Systems (HAIS 2011)*, LNCS, vol. 6679, pp. 155–163, Springer, 2011.

[384] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak: "Ensemble learning for data stream analysis: A survey". *Information Fusion*, vol. 37, pp. 132–156, Elsevier, 2017.

[385] H. M. Gomes, J. P. Barddal, F. Enembreck, A. Bifet: "A survey on ensemble learning for data stream classification". *ACM Computing Surveys*, vol. 50(2), pp. 23:1–23:36, ACM, 2017.

[386] G. Krempl: "The algorithm APT to classify in concurrence of latency and drift". In: J. Gama, E. Bradley, J. Hollmén (eds.) *Advances in Intelligent Data Analysis X (IDA 2011)*, LNCS, vol. 7014, pp. 222–233, Springer, 2011.

[387] G. Krempl, V. Hofer: "Classification in presence of drift and latency". In: M. Spiliopoulou, H. Wang, D. Cook, J. Pei, W. Wang, O. Zaïane, X. Wu (eds.) *The 2011 IEEE 11th International Conference on Data Mining Workshops (ICDMW 2011)*, pp. 596–603, IEEE, 2011.

[388] V. Hofer, G. Krempl: "Drift mining in data: A framework for addressing drift in classification". *Computational Statistics & Data Analysis*, vol. 57(1), pp. 377–391, Elsevier, 2013.

[389] R. Alaiz-Rodríguez, A. Guerrero-Curieses, J. Cid-Sueiro: "Class and subclass probability re-estimation to adapt a classifier in the presence of concept drift". *Neurocomputing*, vol. 74(16), pp. 2614–2623, Elsevier, 2011.

[390] J. Sarnelle, A. Sanchez, R. Capo, J. Haas, R. Polikar: "Quantifying the limited and gradual concept drift assumption". In: *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN'15)*, pp. 1–8, IEEE, 2015.

[391] V. M. A. Souza, D. F. Silva, G. E. A. P. A. Batista, J. Gama: "Classification of evolving data streams with infinitely delayed labels". In: *14th IEEE International Conference on Machine Learning and Applications (ICMLA 2015)*, pp. 214–219, IEEE, 2015.

[392] G. Giacinto, F. Roli: "Intrusion detection in computer networks by multiple classifier systems". In: R. Kasturi, D. Laurendeau, C. Suen (eds.) *Proceedings of the 16th International Conference on Pattern Recognition*, vol. 2, pp. 390–393, IEEE, 2002.

[393] S. Chebrolu, A. Abraham, J. P. Thomas: "Feature deduction and ensemble design of intrusion detection systems". *Computers & Security*, vol. 24(4), pp. 295–307, 2005.

[394] G. Folino, P. Sabatino: "Ensemble based collaborative and distributed intrusion detection systems: A survey". *Journal of Network and Computer Applications*, vol. 66, pp. 1–16, Elsevier, 2016.

[395] G. M. Weiss: "Mining with rarity: A unifying framework". *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets*, vol. 6(1), pp. 7–19, ACM, 2004.

[396] S. García, F. Herrera: "Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy". *Evolutionary Computation*, vol. 17(3),

pp. 275–306, MITP, 2009.

[397] B. Krawczyk: "Learning from imbalanced data: Open challenges and future directions". *Progress in Artificial Intelligence*, vol. 5(4), pp. 221–232, Springer, 2016.

[398] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse: "Hybrid sampling for imbalanced data". In: *Proceedings of the 2008 IEEE International Conference on Information Reuse and Integration (IEEE IRI-2008)*, pp. 202–207, IEEE, 2008.

[399] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu: "A framework for clustering evolving data streams". In: J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, A. Heuer (eds.) *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003)*, pp. 81–92, Morgan Kaufmann, 2003.

[400] T. Zhang, R. Ramakrishnan, M. Livny: "BIRCH: An efficient data clustering method for very large databases". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, pp. 103–114, ACM, 1996.

[401] F. Cao, M. Ester, W. Qian, A. Zhou: "Density-based clustering over an evolving data stream with noise". In: *Proceedings of the 6th SIAM Conference on Data Mining (SDM 2006)*, pp. 328–339, SIAM, 2006.

[402] M. Ester, H.-P. Kriegel, J. Sander, X. Xu: "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pp. 226–231, AAAI Press, 1996.

[403] Y. Chen, L. Tu: "Density-based clustering for real-time stream data". In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pp. 133–142, ACM, 2007.

[404] M. Hahsler, M. Bolaños: "Clustering data streams based on shared density between micro-clusters". *IEEE Transactions on Knowledge and Data Engineering*, vol. 28(6), pp. 1449–1461, IEEE, 2016.

[405] M. Markou, S. Singh: "Novelty detection: A review - Part 1: Statistical approaches". *Signal processing*, vol. 83(12), pp. 2481–2497, Elsevier, 2003.

[406] Y.-H. Liu, Y.-C. Liu, Y. Chen: "Fast support vector data descriptions for novelty detection". *IEEE Transactions on Neural Networks*, vol. 21(8), pp. 1296–1313, IEEE, 2010.

[407] C. Gruhl, B. Sick, A. Wacker, S. Tomforde, J. Hähner: "A building block for awareness in technical systems: Online novelty detection and reaction with an application in intrusion detection". In: *The IEEE 7th International Conference on Awareness Science and Technology (iCAST 2015)*, pp. 194–200, IEEE, 2015.

[408] C. Giraud-Carrier: "A note on the utility of incremental learning". *AI Communications*, vol. 13(4), pp. 215–223, IOS Press, 2000.

[409] S. Lange, G. Grieser: "On the power of incremental learning". *Theoretical Computer Science*, vol. 288(2), pp. 277–307, Springer, 2002.

[410] A. Gepperth, B. Hammer: "Incremental learning algorithms and applications". In: *24th European Symposium on Artificial Neural Networks (ESANN 2016)*, pp. 357–368, 2016.

[411] F. Stahl, M. Gaber, M. M. Salvador: "A modular adaptive classification rule learning algorithm for data streams". In: M. Bramer, M. Petridis (eds.) *Research and Development in Intelligent Systems XXIX (SGAI 2012)*, pp. 65–78, Springer, 2012.

[412] T. Le, F. Stahl, J. B. Gomes, M. M. Gaber, G. Di Fatta: "Computationally efficient rule-based classification for continuous streaming data". In: M. Bramer, M. Petridis (eds.) *Research and Development in Intelligent Systems XXXI (SGAI 2014)*, pp. 21–34, Springer, 2014.

[413] D. Angluin, P. Laird: "Learning from noisy examples". *Machine Learning*, vol. 2(4), pp. 343–370, Kluwer, 1988.

[414] D. Gimlin: "A parametric procedure for imperfectly supervised learning with unknown class probabilities". *IEEE Transactions on Information Theory*, vol. 20(5), pp. 661–663, IEEE, 1974.

[415] R. Barandela, E. Gasca: "Decontamination of training samples for supervised pattern recognition methods". In: F. J. Ferri, J. M. Iñesta, A. Amin, P. Pudil (eds.) *Advances in Pattern Recognition (SSPR/SPR 2000)*, LNCS, vol. 1876, pp. 621–630, Springer, 2000.

[416] B. Frénay, A. Kabán: "A comprehensive introduction to label noise". In: *22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2014)*, pp. 357–368, 2016.

[417] P. Domingos: "Efficient specific-to-general rule induction". In: E. Simoudis, J. Han, U. Fayyad (eds.) *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pp. 319–322, AAAI Press, 1996.

[418] P. Kosina, J. Gama: "Handling time changing data with adaptive very fast decision rules". In: P. A. Flach, T. De Bie, N. Cristianini (eds.) *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2012)*, LNCS, vol. 7523, pp. 827–842, Springer, 2012.

[419] G. Widmer, M. Kubat: "Learning in the presence of concept drift and hidden contexts". *Machine Learning*, vol. 23(1), pp. 69–101, Kluwer, 1996.

[420] M. A. Maloof: "Incremental rule learning with partial instance memory for changing concepts". In: *Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN'03)*, vol. 4, pp. 2764–2769, 2003.

[421] F. J. Ferrer-Troyano, J. S. Aguilar-Ruiz, J. C. Riquelme: "Incremental rule learning and border examples selection from numerical data streams". *Journal of Universal Computer Science*, vol. 11(8), pp. 1426–1439, 2005.

[422] P. Domingos, G. Hulten: "Mining high-speed data streams". In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pp. 71–80, ACM, 2000.

[423] W. Hoeffding: "Probability inequalities for sums of bounded random variables". *Journal of the American Statistical Association*, vol. 58(301), pp. 13–30, Taylor & Francis, 1963.

[424] P. Kosina, J. Gama: "Very fast decision rules for multi-class problems". In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC'12)*, pp. 795–800, ACM, 2012.

[425] J. Gama, R. Rocha, P. Medas: "Accurate decision trees for mining high-speed data streams". In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pp. 523–528, ACM, 2003.

[426] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer: "MOA: Massive online analysis". *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.

[427] M. Deckert, J. Stefanowski: "RILL: Algorithm for learning rules from streaming data with concept drift". In: T. Andreasen, H. Christiansen, J. C. Cubero, Z. W. Raś (eds.) *Foundations of Intelligent Systems (ISMIS 2014)*, pp. 20–29, Springer, 2014.

[428] M. Deckert: "Incremental rule-based learners for handling concept drift: An overview". *Foundations of Computing and Decision Sciences*, vol. 38(1), pp. 35–65, 2013.

[429] F. Beer, U. Bühler: "Learning adaptive decision rules inside relational database systems". In: *Proceedings of the 2nd International Symposium of Fuzzy and Rough Sets (ISFUROS 2017)*, pp. 41:1–41:12, 2017.

[430] F. Beer, U. Bühler: "In-database rule learning under uncertainty: A variable precision rough set approach". In: R. Bello, R. Falcon, J. L. Verdegay (eds.) *Uncertainty Management with Fuzzy and Rough Sets - Recent Advances and Applications*, STUDFUZZ, vol. 377, pp. 257–287, Springer, 2019.

[431] J. R. Quinlan: "Learning logical definitions from relations". *Machine Learning*, vol. 5(3), pp. 239–266, Kluwer, 1990.

[432] S. Muggleton: "Inductive logic programming". *New Generation Computing*, vol. 8(4), pp. 295–318, Springer, 1991.

[433] N. Lavrač, P. Flach, B. Zupan: "Rule evaluation measures: A unifying view". In: S. Džeroski, P. Flach (eds.) *Inductive Logic Programming (ILP-99)*, LNCS, vol. 1634, pp. 174–185, Springer, 1999.

[434] Ł. Wróbel, M. Sikora, M. Michalak: "Rule quality measures settings in classification, regression and survival rule induction - An empirical approach". *Fundamenta Informaticae*, vol. 149(4), pp. 419–449, IOS Press, 2016.

[435] A. Bifet, R. Gavaldà: "Learning from time-changing data with adaptive windowing". In: C. Apte, B. Lui, S. Parthasarathy, D. Skillicorn (eds.) *Proceedings of the 7th SIAM International Conference on Data Mining (SDM 2007)*, pp. 443–448, SIAM, 2007.

[436] R. Klinkenberg, T. Joachims: "Detecting concept drift with support vector machines". In: *Proceedings of the 17th International Conference on Machine Learning (ICML'00)*, pp. 487–494, Morgan Kaufmann, 2000.

[437] M. Datar, A. Gionis, P. Indyk, R. Motwani: "Maintaining stream statistics over sliding windows". In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pp. 635–644, SIAM, 2002.

[438] M. A. Maloof, R. S. Michalski: "Selecting examples for partial memory learning". *Machine Learning*, vol. 41(1), pp. 27–52, Kluwer, 2000.

[439] G. Krempl, I. Žliobaitė, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, J. Stefanowski: "Open challenges for data stream mining research". *ACM SIGKDD Explorations Newsletter - Special*

Issue on Big Data, vol. 16(1), pp. 1–10, ACM, 2014.

[440] T. R. Hoens, R. Polikar, N. V. Chawla: "Learning from streaming data with concept drift and imbalance: An overview". *Progress in Artificial Intelligence*, vol. 1(1), pp. 89–101, Springer, 2012.

[441] Q. Zhu, X. Hu, Y. Zhang, P. Li, X. Wu: "A double-window-based classification algorithm for concept drifting data streams". In: X. Hu, T. Y. Lin, V. Raghavan, J. Grzymala-Busse, Q. Liu, A. Broder (eds.) *Proceedings of the 2010 IEEE International Conference on Granular Computing (GrC 2010)*, pp. 639–644, IEEE, 2010.

[442] E. S. Xioufis, M. Spiliopoulou, G. Tsoumakas, I. Vlahavas: "Dealing with concept drift and class imbalance in multi-label stream classification". In: T. Walsh (ed.) *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, vol. 2, pp. 1583–1588, AAAI Press, 2011.

[443] J. Fürnkranz: "Separate-and-conquer rule learning". *Artificial Intelligence Review*, vol. 13, pp. 3–54, Kluwer, 1999.

[444] Y. Yao: "Three-way decision: An interpretation of rules in rough set theory". In: P. Wen, L. Polkowski, Y. Yao, S. Tsumoto, G. Wang (eds.) *Rough Sets and Knowledge Technology (RSKT 2009)*, LNCS, vol. 5589, pp. 642–649, Springer, 2009.

[445] R. Elmasri, S. B. Navathe: "Fundamentals of database systems". 6th edition, Addison-Wesley, 2010.

[446] M. B. Roeser: "Oracle® Database SQL language reference (12c release 1)", Oracle Corp., `https://docs.oracle.com/database/121/SQLRF/E41329-25.pdf`, 2017.

[447] E. Ikonomovska, J. Gama, S. Džeroski: "Learning model trees from evolving data streams". *Data Mining and Knowledge Discovery*, vol. 23(1), pp. 128–168, Springer, 2011.

[448] M. Harries: "Splice-2 comparative evaluation: Electricity pricing", University of New South Wales (Sydney, Australia), Technical Report, 1999.

[449] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, R. Gavaldà: "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, pp. 139–148, ACM, 2009.

[450] V. Losing, B. Hammer, H. Wersing: "Interactive online learning for obstacle classification on a mobile robot". In: *Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN'15)*, pp. 1–8, 2015.

[451] G. Hulten, L. Spencer, P. Domingos: "Mining time-changing data streams". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pp. 97–106, ACM, 2001.

[452] C. J. van Rijsbergen: "Foundations of evaluation". *Journal of Documentation*, vol. 30(4), pp. 365–373, MCB UP, 1997.

[453] C. D. Manning, P. Raghavan, H. Schütze: "Introduction to information retrieval". Cambridge University Press, 2012.

[454] P. A. Dawid: "Present position and potential developments: Some personal views:

statistical theory: the prequential approach". *Journal of the Royal Statistical Society*, vol. 147(2), pp. 278–292, Wiley, 1984.

[455] J. Gama, R. Sebastião, P. Pereira Rodrigues: "Issues in evaluation of stream learning algorithms". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, pp. 329–338, ACM, 2009.

[456] M. Friedman: "The use of ranks to avoid the assumption of normality implicit in the analysis of variance". *Journal of the American Statistical Association*, vol. 32(200), pp. 675–701, Taylor & Francis, 1937.

[457] F. Wilcoxon: "Individual comparisons by ranking methods". *Biometrics Bulletin*, vol. 1(6), pp. 80–83, Wiley, 1945.

[458] J. Demšar: "Statistical comparisons of classifiers over multiple data sets". *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[459] J. Stefanowski, D. Vanderpooten: "Induction of decision rules in classification and discovery-oriented perspectives". *International Journal of Intelligent Systems*, vol. 16(1), pp. 13–27, Wiley, 2001.

[460] K. McGarry: "A survey of interestingness measures for knowledge discovery". *The Knowledge Engineering Review*, vol. 20(1), pp. 39–61, Cambridge University Press, 2005.

[461] L. Geng, H. Hamilton: "Interestingness measures for data mining: A survey". *ACM Computing Surveys*, vol. 38(3), pp. 9:1–9:32, ACM, 2006.

[462] P. Kosina: "Decision rule learning for evolving data streams", Masaryk University (Brno, Czech Republic), PhD Thesis, 2013.

[463] "Best practices for implementing high volume IoT workloads with Oracle database 12c", Oracle Corp., `https://www.oracle.com/a/tech/docs/wp-bp-for-iot-with-12c-042017-3679918.pdf`, 2017.

[464] "Distributed runtime environment", Apache Flink, `https://ci.apache.org/projects/flink/flink-docs-release-1.4/concepts/runtime.html`, 2018.

[465] "Operators", Apache Flink, `https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/`, 2018.

[466] "Windows", Apache Flink, `https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/windows.html`, 2018.

[467] S. García, J. Luengo, F. Herrera: "Data preprocessing in data mining", Intelligent Systems Reference Library, vol. 72. Springer, 2015.

[468] "Oracle® TimesTen Application-Tier Database Cache: Introduction (11g release 2)", Oracle Corp., `https://docs.oracle.com/cd/E11882_01/timesten.112/e21631.pdf`, 2014.

[469] F. Faerber, A. Kemper, P.-A. Larson, J. Levandoski, T. Neumann, A. Pavlo: "Main memory database systems". *Foundations and Trends in Databases*, vol. 8(1-2), pp. 1–130, 2017.

[470] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, M. Zwilling: "Hekaton: SQL Server's memory-optimized OLTP engine". In: K. A. Ross, D. Srivastava, D. Papadias (eds.) *Proceedings of the 2013*

ACM SIGMOD International Conference on Management of Data (SIGMOD'13), pp. 1243–1254, ACM, 2013.

[471] K. Stolze, G. Lohman, V. Raman, R. Sidle, F. Beier: "Evolutionary integration of in-memory database technology into IBM's enterprise DB2 database systems". In: M. Horbach (ed.) *Informatik 2013 - 43. Jahrestagung der Gesellschaft für Informatik*, LNI, vol. 220, pp. 461–471, GI, 2013.

[472] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T.-H. Lee, N. Macnaughton, V. Marwah, N. Mukherjee, A. Mullick, S. Muthulingam, V. Raja, M. Roth, E. Soylemez, M. Zait: "Oracle database in-memory: A dual format in-memory database". In: J. Gehrke, W. Lehner, K. Shim, S. K. Cha, G. M. Lohman (eds.) *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering (ICDE'15)*, pp. 1253–1258, IEEE, 2015.

[473] "Oracle® TimesTen in-memory database: Operations guide (11g release 2)", Oracle Corp., `https://docs.oracle.com/cd/E11882_01/timesten.112/e21633.pdf`, 2014.

[474] "Oracle® TimesTen in-memory database: Reference (11g release 2)", `https://docs.oracle.com/cd/E21901_01/timesten.1122/e21643.pdf`, Oracle Corp., 2015.

[475] "Oracle® TimesTen Application-Tier Database Cache: User's guide (11g release 2)", Oracle Corp., `https://docs.oracle.com/cd/E21901_01/timesten.1122/e21634.pdf`, 2014.

[476] C. Pape, R. Trommer, S. Rieger: "Energieverbrauch von Live-Migrationen in OpenStack-basierten Private-Cloud-Umgebungen". In: P. Müller, B. Neumair, H. Reiser, G. Dreo Rodosek (eds.) *8. DFN-Forum Kommunikationstechnologie*, LNI, vol. P-243, pp. 127–136, GI, 2015.

[477] K. Spindler, S. Reißmann, R. Trommer, C. Pape, S. Rieger, T. Glotzbach: "AEQUO: Enhancing the energy efficiency in private clouds using compute and network power management functions". *International Journal on Advances in Internet Technology*, vol. 8(1–2), pp. 13–28, 2015.

[478] J. Gao, W. Fan, J. Han: "On appropriate assumptions to mine data streams: Analysis and practice". In: N. Ramakrishnan, O. R. Zaïane, Y. Shi, C. W. Clifton, X. Wu (eds.) *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007)*, pp. 143–152, IEEE, 2007.

[479] X. Zhang, W. Wei: "Self-adaptive change detection in streaming data with non-stationary distribution". In: L. Cao, Y. Feng, J. Zhong (eds.) *Advanced Data Mining and Applications (ADMA 2010)*, LNCS, vol. 6440, pp. 334–345, Springer, 2010.

[480] G. W. Milligan: "A monte carlo study of thirty internal criterion measures for cluster analysis". *Psychometrika*, vol. 46, pp. 187–199, 1981.

[481] A. Rosenberg, J. Hirschberg: "V-measure: A conditional entropy-based external cluster evaluation measure". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pp. 410–420, 2007.

[482] J. Wu, H. Xiong, J. Chen: "Adapting the right measures for k-means clustering". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*, pp. 877–886, ACM, 2009.

[483] A. Bifet, R. Gavaldà: "Adaptive learning from evolving data streams". In: N. M. Adams, C. Robardet, A. Siebes, J. F. Boulicaut (eds.) *Advances in Intelligent Data Analysis VIII*, LNCS, vol. 5772, pp. 249–260, Springer, 2009.

[484] "Replication", Apache Kafka, `https://kafka.apache.org/0100/documentation.html#replication`, 2016.

[485] "Data streaming fault tolerance", Apache Flink, `https://ci.apache.org/projects/flink/flink-docs-release-1.4/internals/stream_checkpointing.html`, 2018.

[486] S. Kokoska, C. Nevison: "Statistical tables and formulae". STS, Springer, 1989.

[487] G. W. Corder, D. I. Foreman: "Nonparametric statistics: A step-by-step approach". 2nd edition, Wiley, 2014.

ISBN 978-3-7376-1059-9