Generative Pattern Dissemination for Collaborative Intrusion Detection

A thesis presented for the degree of Master of Science
by Mike Petersen

First Reviewer: Prof. Dr. Ulrich Bühler
Second Reviewer: Prof. Dr. Sebastian Rieger

Department of Computer Sciences
University of Applied Sciences Fulda
Fulda, Germany

July 2021

Abstract

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 Objectives

1.3 Structure

Chapter 2

Preliminaries

2.1 Intrusion Detection

This section introduces the topic of intrusion detection and some important subcategories in the context of the thesis. First, Intrusion Detection Systems (IDSs) are described and categorized according to different characteristics. The advantages and disadvantages of individual types of IDS are explained and examples are referred to individually. The topic of network flow monitoring is then addressed, since it is an essential component of network monitoring that is incorporated in the proposed architecture. The role of network flows in the context of network monitoring and management is motivated and the typical processes of a flow exporter are described. In the context of the shortcomings of conventional IDSs for protecting large scale systems against coordinated and distributed attacks Collaborative Intrusion Detection Systems (CIDSs) are introduced. For this purpose, it is first described what coordinated and distributed attacks are. Then it is explained what constitutes a CIDS, what requirements exist for the realization of CIDS, and what components and processes CIDS typically consist of.

2.1.1 Intrusion Detection Systems

Classic security mechanisms, such as encryption or firewalls, are considered as preventive measures for protecting IT infrastructures. However, in order to be able to react to security breaches that have already occurred, additional reactive mechanisms are required. To complement preventive measures, IDSs have been commercially available since the late 1990s [WM18, p. 27]. Generally, the main reason for operating an IDS is to monitor and analyze computer networks or systems in order to identify anomalies, intrusions or privacy violations [Hin+20]. Specifically, the following three significant advantages are pointed out [WM18, p. 391].

- IDSs can detect the preliminaries of attacks, in particular the organized gathering of information about networks and defense mechanisms (attack reconnaissance), and thus enable the prevention or mitigation of damage to information assets.
- IDSs can help protect information assets when known vulnerabilities cannot be fixed fast enough, notably in the context of an rapidly changing threat environment.
- The occurrence of unknown security vulnerabilities (zero day vulnerabilities) is not predictable, meaning that no specific preparations can be made for them. However, IDSs can identify processes in the IT system that deviate from the normal state and thus contribute to the detection of zero day attacks

For an effective IDS, it is important to be able to detect as many steps as possible within the prototypical attack sequence, also called kill chain [WM18, p. 393]. Since a successful intrusion into a system can be stopped at several points in this sequence, the effectiveness of the IDS increases with its functionality. The following categorization (see Figure 2.1) of intrusion attempts according to [Ken99] reflects parts of that chain.

Probing refers to the preambles of actual attacks, also known as attack reconnaissance. This includes obtaining information about an organization and its network behavior (footprinting) and obtaining detailed information about the used operating systems, network protocols or hardware devices (fingerprinting).

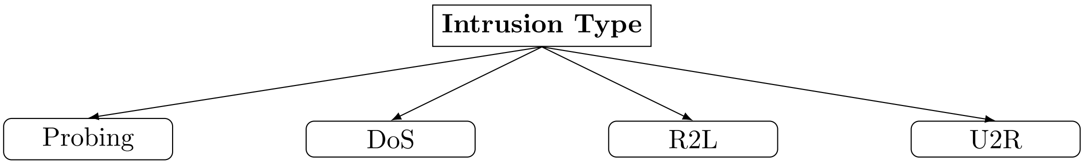


Figure 2.1: A categorization of intrusions into different subtypes

Denial of Service (DoS) refers to an attack aimed at disabling a particular service for legitimate users by overloading the target systems processing capacity. Remote to Local (R2L) attacks attempt to gain local access to the target system via a network connection. And one step further, User to Root (U2R) attacks start out with user access on the system and gain root access by exploiting vulnerabilities. Individual intrusion types can but do not have to be executed consecutively. For example, it is not mandatory to render a system incapable of action by DoS in order to subsequently gain local access to the target. However, this may be part of a strategy.

Additionally, IDSs are generally categorized by the detection scope and the employed attack detection method [Mil+15]. Hence, a distinction is made between Host-based Intrusion Detection Systems (HIDSs) and Network-based Intrusion Detection Systems (NIDSs). While a HIDS resides on a system, known as host, only monitoring local activities, a NIDS resides on a network segment and monitors remote attacks that are carried out across the segment. In general, HIDS are advantageous if individual hosts are to be monitored. NIDS, on the other hand, are able to monitor traffic coming in and out of several hosts simultaneously. The disadvantage of NIDS, however, is that attacks can only be detected if they are also reflected in the network traffic. Pioneering examples in the context of HIDS are the Intrusion-Detection Expert System (IDES) [LTG92] and the Multics Intrusion Detection and Alerting System (MIDAS) [Seb+88]. A prominent representative in the area of NIDS is NetSTAT [VK98] [VK99].

Furthermore, IDSs are categorized into misuse-based detection and anomaly-based detection (see Figure 2.2). A misuse-based approach defines a model that describes intrusive behaviour and compares the system or network state against that model. An anomaly-based IDS, on the other hand, creates a statistical baseline profile of the system's or network's normal state and compares it with monitored activities. The concept of the NIDS origins from [Den87].

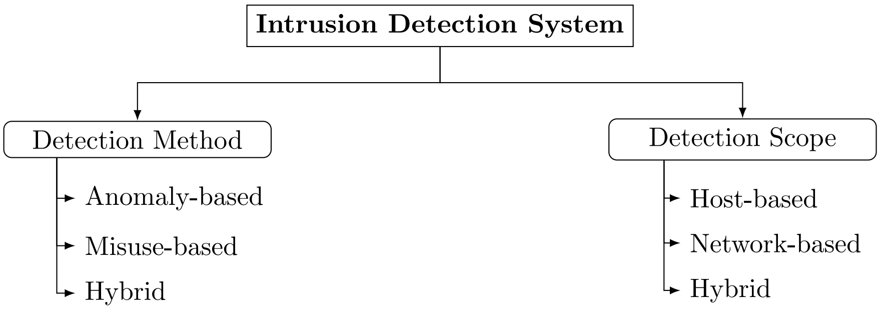


Figure 2.2: A taxonomy for IDS that distincts systems based on the utilized detection method or the scope in which it operates

Obviously, these two opposing approaches offer different advantages and disadvantages and are capable of complementing each other. Misuse-based systems can detect malicious behavior with a low false positive rate, but assume that the exact patterns are known. Typically, this method cannot detect novel [WM18, p. 403], metamorphic or polymorphic attacks [Szo05, p. 236]. An anomaly-based approach allows both known and unknown attacks to be detected, but the frequent occurrence of false-positive estimations is a major challenge. Given the amount of data that occurs in computer systems and networks nowadays, the generation of false alarms for even a fraction of this amount can render the IDS operationally unusable, since no network administrator can investigate such a large number of incidents in detail [Axe00]. Most implementations of misuse-based systems rely on the creation of signatures. A signature is similar to a collection of rules that describes an attack pattern. The most prominent signature-based IDS is Snort [Roe+99]. The payload-based anomaly detector PAYL [WS04] is an example for an anomaly-based system. Hybrid approaches are conceivable in order to circumvent the respective disadvantages of different subcategories of IDSs. The interested reader is referred to the following works [Dep+05] [ZZ06] [Bee21].

2.1.2 Network Flow Monitoring

Network monitoring is a key component for network management as network administrators derive decisions based on data that results from monitoring activities. By collecting data from network devices, such as switches, routers or clients, the current behaviour of the network is measured. Commonly, this behaviour has to meet some minimum application requirements, which can be summarized by the term Quality of Service (QoS) [TFW21, p. 406]. In this context, the most early and vague definition of a network flow was introduced as a sequence of packets traveling from the source to the destination [Cla88]. Here, no further characteristics of a flow are given and it is not clear how one flow is differentiated from another flow. Later in the 1990s, however, the demand arose for a finer-grained view of network traffic than that provided by interface-level counters, but without the disadvantage of the huge amounts of data generated by packet tracking. Evidently, the concept of network flows has filled this gap in demand and has gained a central role in network management and research today [TB11].

Since the first attempts at standardization in the 1990s through the efforts of the Internet Enginnering Task Force (IETF), a number of proprietary standards have emerged as each network device vendor has implemented its own flow export protocol. In order to improve interoperability in the area of network flow measurement, the IETF, established the IP Flow Information Export (IPFIX) working group. This working group defined the IPFIX standard (RFC 5101), which defines the term network flow as follows.

Definition 1 (Network Flow)**.** A network flow is an aggregation of all network packets that pass an observation point of a network during a certain time interval and share a set of common properties. These properties are defined as the result of applying a function to the value of

1. one or more packet, transport or application header fields,
2. one ore more characteristics of the packet itself,
3. one or more fields derived from packet treatment.

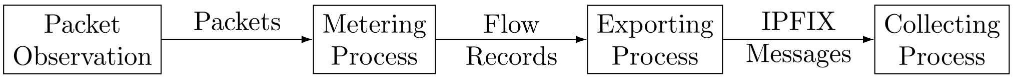


Figure 2.3: General Flow Monitoring Architecture

This definition is loose enough to cover the range from a flow containing all packets observed at a network interface to a flow that consists only of a single packet between two applications. Note, that this definition of the set of properties is also less strict than the conventional definition of the five-tuple consisting of source IP address, source port, destination IP address, destination port and protocol [Cla08].

In general, network flows can appear either as unidirectional flow, which aggregates all packets from one host to another host, or as bidirectional format, that aggregates all packets regardless of direction. Depending on which flow format and flow exporter is used, additional information, e.g. statistical calculations on bytes per second, can be obtained. Other common network flow protocols are NetFlow [Cla04], OpenFlow [McK+08] and sFlow [Pha].

Systems that generate flow data and extract additional information are called flow exporters. Generally, flow exporters are part of a general flow monitoring architecture [Hof+14], that consists of four different processes (see Figure 2.3). The packet observation is done on interfaces of packet forwarding devices by capturing network packets and executing specific preprocessing steps like timestamping and truncation. The process of flow metering describes the aggregation of packets into flow records. There, a set of functions, including packet header capturing, timestamping, sampling and classifying is applied on values of the packet stream listed above. Furthermore, flow records are maintained, which may include creating, deleting or udpating records, or computing statistical flow features. After that, the exporting process places flow records in a datagram of the deployed export protocol and sends them to one or more collecting processes. Finally, the collection process is responsible for the reception, storage and preprocessing of flow records, produced by the preceding step. Typically, the collected data is analyzed. This can be done, for example, in the context of an IDS.

Using network flows for intrusion detection has several advantages over Deep Packet Inspection (DPI). For example, DPI requires highly complex and expensive infrastructure for storage and analysis of the data [Hof+14]. Since flow exporter setups rely on packet header fields and statistical aggregations, a significant data reduction is achieved, which is why an analysis of network flows is scalable and also applicable for high-speed networks [Hof+14]. In addition, discarding the payload results in a better compliance with data retention laws and privacy policies.

2.1.3 Collaborative Intrusion Detection

With the commercialization of cloud products, various aspects related to the security of computer networks have also become more stringent. Not only has part of the responsibility for securing servers and networks shifted from the end customer to the service provider. The fact that computing resources within large data centers that are operated by public cloud providers are publicly accessible offers, on the one hand, a large contiguous attack surface. And on the other hand, a large amount of potential resources for executing so-called large-scale coordinated attacks are accessible to attackers.

Typically, large-scale coordinated attacks target a large number of hosts which are spread over a wide geographical area [ZLK10]. There, attackers make use of automation and sophisticated tools to target all vulnerable services at once instead of manually targeting services [Sav05]. In the context of cloud environments, attackers can hijack mismanaged user accounts and hack into poorly secured cloud deployments [KG19]. In this way, attackers can gain access to a variety of different resources to spread attacks across as many sources as possible, thus avoiding detection by security systems.

According to [ZLK10], large-scale coordinated attacks are difficult to detect by isolated IDS, because of their limited monitoring scope. For instance, in [RGH12] the authors show that a distributed portscan can be executed with relatively few resources without being detected when deploying Snort IDS and a commercial firewall solution. The experimental results show that the detection by the IDS can be bypassed most effectively by distributing the sources of a scan, whereas the detection by the firewall can be evaded by slowing down single executions of scans.

In order to detect large-scale coordinated attacks, evidence of intrusions from multiple different sources need to be combined. This idea leads to the development of CIDS, where the aggregation and correlation of data originating from different IDSs creates a holistic picture of the network to be monitored and enables the detection of distributed and coordinated attacks [Vas16, p. 24]. CIDS are also a solution to zero day attacks, because information about novel attacks can be shared with participants of the CIDS in order to mitigate such attacks at an early stage. In addition to the aspect of improved detection, another advantage is the improved scaling for the protection of large IT systems. CIDSs can monitor large-scale networks more effectively with the realization of a loadbalancing strategy. By sharing IDS resources across different infrastructures peak loads can be served resulting in a reduced downtime of individual IDSs.

In general, a CIDS is defined as a network of several intrusion detection components that collect and exchange data on system security. The tasks of a CIDS can be allocated across two main components, namely the monitoring units and the analysis units. Monitoring units can be considered as conventional IDS that monitor a sub-network or a host and by that, generate low-level intrusion alerts. Analysis units are responsible for merging the low-level intrusion alerts and their further post-processing. This includes, for instance, the correlation of the alerts, the generation of reports or the distribution of the information to the participants of the network. The communication between the monitoring and analysis units is largely determined by the architecture of the CIDS.

CIDS architectures can be categorized into centralized, hierarchical and decentralized approaches [ZLK10] (see Figure 2.4). Centralized architectures [CM02][MI03], consisting of multiple monitoring units and a central analysis unit, suffer from a conceptual single point of failure (SPoF) and are limited in their scalability due to a bottleneck, which is introduced by the central analysis unit. However, in practice the SPoF can be avoided if individual components are implemented redundantly and form a centralized architecture only at the logical level. A bottleneck, on the other hand, is usually avoided by a distributed implementation of the logically centralized architecture. Hierarchical designs exhibit multiple monitoring and analysis units organized in a tree-based topology [PN97] [Zha+01] [Ngu+19]. These systems are restricted in their scalability by the respective instances on higher levels, whose failure results in a malfunction of the respective sub-trees [ZLK10].

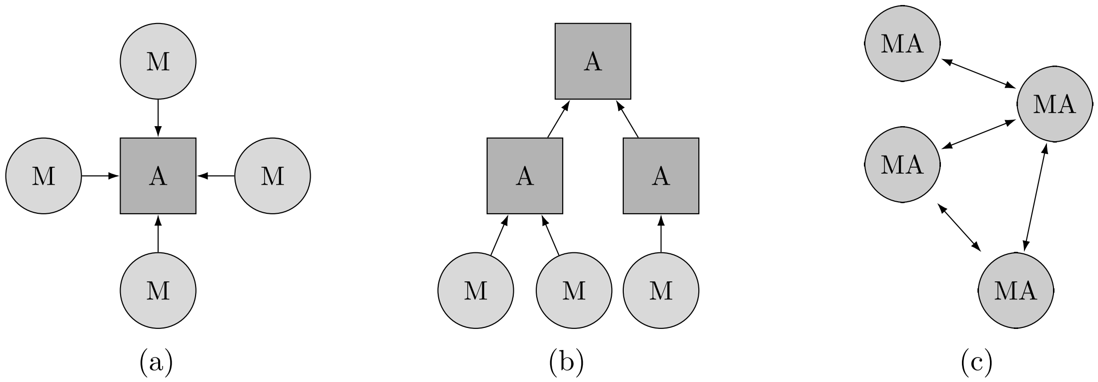


Figure 2.4: CIDS architectures: (a) hierarchical, (b) hierarchical or (c) distributed layout of monitoring (M) and aggregation units (A)

Again, such disadvantages only play a major role in non-redundantly implemented systems. Additionally, there exist approaches that are inherently distributed. Distributed architectures [JWQ03] [Fun+08] [Vas+15], wherein each participating system consists of both a monitoring and analysis unit and where the communication is based on some form of data distribution protocol, are considered as scalable by design. However, they depend on effective and consistent data dissemination and the data attribute selected for correlation may affect load distribution among participants [ZLK10].

The authors in [Vas+15] describe the most important requirements that a CIDS has to meet. Additionally, they further categorize the components of a CIDS by function. The following is a summary of these requirements and components. Six requirements are defined, namely accuracy, minimal overhead and scalability, resilience, privacy, self-configuration and lastly interoperability. In this context, accuracy generally describes a collection of evaluation metrics for assessing the performance of the CIDS. Frequently used metrics are, for example, the Detection Rate (DR), i.e. the ratio of correctly detected attacks to the total number of attacks, or the False Postive Rate (FPR), which sets the number of normal data classified as an attack in relation to the total number of normal data. Obviously, a CIDS is expected to have higher accuracy in terms of attack detection than isolated IDSs. The requirement for minimal overhead and scalability addresses the operational overhead and the scalability of the system. First, the algorithms and techniques for collecting, correlating, and aggregating data must require minimal computational overhead. Second, data distribution within the CIDS must be efficient. The performance required to operate the system should increase linearly with the resources used, so that computer systems of any size can be protected by the CIDS. This property can be measured in theoretical terms by considering the complexity of the algorithms used. In practical terms, the passed time from the collection of a data point to the decision-making process can be measured. Resilience describes the ability of the system to be resistant to attacks, manipulations and system failures. A distinction is made between external attacks, such as DoS, and internal attacks by infiltrated CIDS components. Moreover, it also covers fail-safety in general, which can be achieved, for example, by avoiding SPoFs.

With regard to the protection and regulation of privacy in the CIDS, a distinction is made between internal and external communication. Data that is exchanged internally in the CIDS network between members may contain potentially sensitive information,

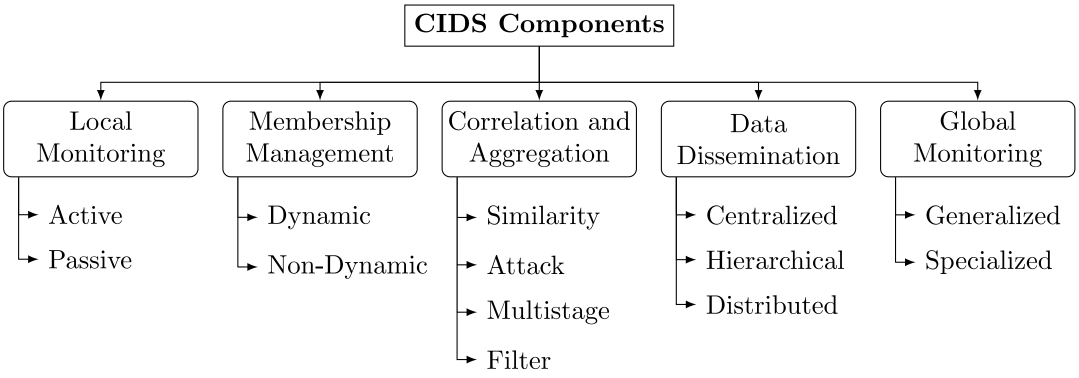


Figure 2.5: Components of CIDS according to functions

Such information should not be disclosed directly to other participants for reasons of data privacy. This includes, among other things, legal aspects when it comes to sharing log and network data. In connection with the resilience of the system, securing communication against third parties using cryptographic methods also plays a role and represents a major challenge, especially in dynamically distributed architectures. Self-configuration describes the degree of automation of a system with regard to configuration and operation. Particularly in distributed and complex architectures, a high degree of automation is desired in order to avoid operating errors and enable automatic resolution of component failures. Finally, individual components of the overall system, which were deployed in different system and network environments, should be able to interact with each other in the context of the CIDS. In addition to system-wide standards for data collection, processing and exchange, there exists a trade-off between interoperability and privacy.

The domains into which the components of the CIDS can be grouped are referred to as local monitoring, membership management, correlation and aggregation, data dissemination, and global monitoring. In the context of local monitoring, a distinction is made between active and passive monitoring. Active monitoring refers to the use of honeypots that reveal themselves as attack targets in the infrastructure in order to collect attack data. Passive monitoring involves intrusion detection activity at the host or network level, which in turn can be divided into misuse-based or anomaly-based methods according to the type of detection technique used (see Section 2.1.1).

Membership management refers to ensuring secure communication channels in the form of an overlay network. Generally, membership management is categorized according to the organization and structure of the system topology. In terms of organization, there are either static approaches, in which members are added or removed manually, or dynamic approaches, in which components are organized automatically via a central entity or a protocol. Furthermore, the structure of the overlay network is relevant for the type of communication in the CIDS network. This means that connections between the monitoring and analysis units are either centralized, hierarchical or distributed.

Before collected and analyzed data are shared with other participants in the CIDS network, the data is correlated and similar data points are aggregated. The main purpose of generating global and synthetic alerts is firstly to reduce the amount of alerts overall and secondly to improve data insights. Correlation mechanisms can be categorized into single-monitor and monitor-to-monitor approaches.

Single-monitor methods correlate data locally without sharing data with other monitor entities. Monitor-to-monitor approaches, on the other hand, share data with other monitoring units in order to correlate local data with shared data and thus enable insights that go beyond isolated methods. Further, a classification can be made according to the correlation techniques used, grouping four different approaches.

Similarity-based approaches correlate data based on the similarity of one or more attributes. For instance, [DW01] suggests using the 5-tuple information of the network data to detect duplicates within NIDSs. The computation of similarity can be done in a variety of ways. For example, [VS01] defines a similarity function for each attribute and computes the overall data similarity by combining the functions using an expectation of similarity. High-dimensional data is usually problematic, as the difficulty for an effective calculation of similarity increases with the number of attributes [ZLK09].

Attack scenario-based approaches detect complex attacks based on databases that provide patterns for attack scenarios [Gar+04] [DC02]. Such approaches have high accuracy for already known attacks. However, the accuracy decreases as soon as the patterns in the real data deviate from those in the database, since the definition of the scenario-rules are of a static quality in these approaches.

Multistage alert correlation aims to detect unknown multistage attacks by correlating defined pre- and post-conditions of individual alerts [CM02] [CLF03]. The idea behind the approach is based on the assumption that an attack is executed in preparation for a next step. The system states before and after an alert are modeled as pre- and postconditions, which are correlated with each other. While these approaches are more flexible than the all-static definition of attack scenarios, they still require the prior modeling of pre- and postconditions, which are based on expert knowledge.

Filter-based approaches filter irrelevant data in order to reduce the number of alerts within the intrusion detection context. For example, [PFV02] filters alerts by a ranking based on priorities of incidents. Priorities are calculated through comparing the target's known topology with the vulnerability requirements of the incident type. The rank that each alert is assigned to provides the probability for its validity. The accuracy of the filters is based on the quality of the description of the topology to be protected and require reconfiguration when changes are made to the infrastructure.

Data Dissemination describes the efficient distribution of correlated and aggregated data in the CIDS network. How the data is disseminated is strongly influenced by the topology and membership management of the system. Centralized topologies have a predefined flow of information from the monitoring units to the central analysis unit. If the system is organized hierarchically, information flows statically or dynamically organized from lower monitoring units in the hierarchy to higher units. Distributed topologies allow the use of versatile strategies, such as flooding, selective flooding or publish-subscribe methods.

Global monitoring mechanisms are needed for the detection of distributed attacks which isolated IDSs cannot detect due to the limited information base. Thus, the detection of distributed attacks relies on the insight obtained by combining data from different infrastructures. This means that global monitoring is strongly dependent on the employed correlation and aggregation techniques. The overall strategy of the CIDS is described by global monitoring and can have both generic and specific objectives.

2.2 Locality Sensitive Hashing

Finding similar objects, formally known as the nearest-neighbour search problem, is a central problem in computer science in general. And besides the exact search, it is also an important topic in the field of intrusion detection. For example, exact searches allow to compare the signatures of known malware with low false positive rates. Considering polymorphic attacks, however, it is essential to also detect all objects that are similar to the known attack and thus detect modified forms of it. In this context, this section presents a well-studied approach called *Locality Sensitive Hashing* (LSH) that solves an approximate version of the nearest-neighbour search problem by partitioning the search space with a hash function. This way, the searching problem is reduced to pairs that are most likely to be similar. Besides the application for solving the NN search problem, for which LSH was originally conceived, the concept has been proven to be effective for numerous other use cases, such as dimensionality reduction, clustering or classification. As the proposed generative pattern database integrates a locality-sensitive hash function for enabling both similarity search and data parallelism, LSH and a specific variant called *Random Projection* (RP) is described in detail in this section.

Section 2.2.1 introduces the approximate version of the nearest neighbour search problem as it is a prerequisite for the general definition of LSH. After that, Section 2.2.2 begins by clarifying the core idea of LSH and subsequently explains how to construct a locality-sensitive hash function. Lastly, a specific family of LSH that uses the cosine distance for similarity calculations, also known as *Random Projection* is presented in Section 2.2.3.

2.2.1 The Approximate Nearest Neigbour Problem

For general definitions of the nearest-neighbour search problem (NN) (see Definition 2) and its variants, a set of points $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}, N \in \mathbb{N}$ in a metric space $(\mathcal{X}, d)$ with $\mathcal{P} \subset \mathcal{X}$ and where d is a metric on $\mathcal{X}$, i.e., a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is considered. It is assumed that d is a proper metric, which means that it is

1. *symmetric*: $d(\boldsymbol{p}, \boldsymbol{q}) = d(\boldsymbol{q}, \boldsymbol{p})$,
2. *reflexive*: $d(\boldsymbol{p}, \boldsymbol{q}) \leq 0$, $d(\boldsymbol{p}, \boldsymbol{q}) = 0 \iff \boldsymbol{p} = \boldsymbol{q}$ and satisfies the
3. *triangle inequality*: $d(\boldsymbol{p}, \boldsymbol{q}) \leq d(\boldsymbol{p}, \boldsymbol{s}) + d(\boldsymbol{s}, \boldsymbol{q})$.

Definition 2 (Nearest-neighbour search problem[IM98])**.** Construct a data structure so as to efficiently answer the following query: Given any query point $\boldsymbol{q} \in \mathcal{X}$, find some point $\boldsymbol{p} \in \mathcal{P}$ such that

$$\min_{\boldsymbol{q} \in \mathcal{X}} d(\boldsymbol{p}, \boldsymbol{q}). \tag{2.1}$$

As the algorithms in the presented approach operate on vectors of statistical flow features (see Section 2.1.2), the input set for specific definitions is usually restricted to $\mathbb{R}^D, D \in \mathbb{N}$. Thus, a specific example of Definition 2 in the context of this work includes high-dimensional data in $\mathcal{P} \in \mathbb{R}^D$ with $D \gg 1$ and could define d as, e.g., the euclidean distance. In this case, an exhaustive search would require a query time of $O(DN)$. Unfortunately, all exact algorithms that provide a better time complexity than an exhaustive search require $O(2^D)$ space [Rub18].

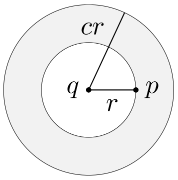


Figure 2.6: In the cr-approximate nearest-neighbour problem some point within cr is accepted, if there exists a point $\boldsymbol{p}$ where $d(\boldsymbol{p}, \boldsymbol{q}) \leq r$.

This tradeoff between time and space complexity is usually referred to as "curse of dimensionality" and can only be resolved by accepting approximate solutions. The c-approximate nearest-neighbour problem (c-ANN) is defined as follows [IM98].

Definition 3 (c-Approximate nearest-neighbour search problem)**.** For any given query point $\boldsymbol{q} \in \mathcal{X}$ and some approximation factor $c > 1$, find some point $\boldsymbol{p} \in \mathcal{P}$ such that

$$d(\boldsymbol{p}, \boldsymbol{q}) < c \cdot \min_{\boldsymbol{s} \in P} d(\boldsymbol{s}, \boldsymbol{q}). \tag{2.2}$$

Thus, the distance from the query point $\boldsymbol{q}$ to the approximate nearest neighbour $\boldsymbol{p}$ is at most c times the distance to the true nearest neighbour $\boldsymbol{s}$. However, LSH does not solve the c-ANN directly. Instead, the problem is relaxed first by introducing the cr-approximate nearest-neighbour problem (cr-ANN) as follows [IM98].

Definition 4 (cr-Approximate nearest-neighbour search problem)**.** For any given query point $\boldsymbol{q} \in \mathcal{X}$, some approximation factor $c > 1$ and some target distance $r > 0$, if there exists a point $\boldsymbol{p} \in \mathcal{P}$ where $d(\boldsymbol{p}, \boldsymbol{q}) \leq r$, then return a point $\boldsymbol{p}' \in \mathcal{P}$ where

$$d(\boldsymbol{p}', \boldsymbol{q}) \leq cr. \tag{2.3}$$

Figure 2.6 depicts the cr-ANN. The target distance r represents the distance of the query object from its nearest neighbour. If there is such a point, the algorithm returns points within cr distance from the query object, else it returns nothing. It is shown that LSH can solve the c-ANN by solving the cr-ANN for different settings of r [IM98].

2.2.2 Locality-Sensitive Hash Functions

Introduced by Indyck and Motwani in 1998 [IM98] as an algorithm that solves the c-ANN, locality-sensitive hashing (LSH) has since been extensively researched and is now considered among the state of the art for approximate searches in high-dimensional spaces.[1] The basic idea of the approach is to partition the input data using a hash function that is sensitive to the location of the input within the metric space. This way, similar inputs collide with a higher probability than inputs that are far apart. Thus, LSH exhibits fundamental differences to conventional hash functions.[2]

A hash function is a function that maps a large input set to a smaller target set. The elements of the input set are called *messages* or *keys* and may be of arbitrary different lengths. The elements of the target set are called *digests* or *hash values* and are of fixed size length.

[1] See [NBJ21] for an exhaustive survey of NN Techniques.

[2] Cryptographic and non-cryptograhpic hash functions are referred to as conventional hashing.

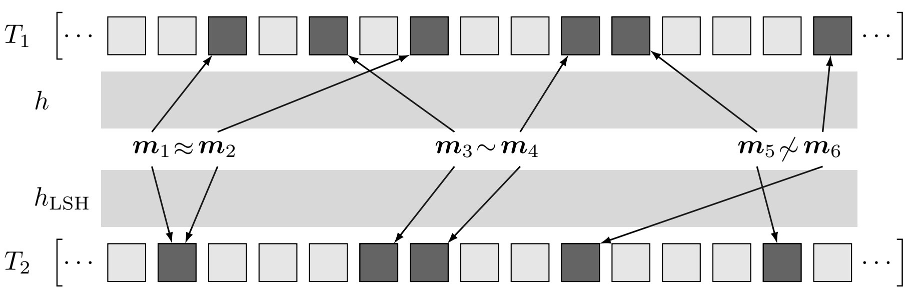


Figure 2.7: The behaviour of a conventional hashing function h and a locality-sensitive hashing function h_{LSH} when hashing very similar messages $\boldsymbol{m} \approx \boldsymbol{m}$, similar messages $\boldsymbol{m} \sim \boldsymbol{m}$ and non-similar messages $\boldsymbol{m} \not\sim \boldsymbol{m}$ into the slots of a hash table T_1 and T_2 respectively.

We define a conventional hash function as $h : \mathcal{M} \to \mathcal{B}$ where $\mathcal{M} \subset \mathcal{X}$ is the input set and $\mathcal{B} = \{0,1\}^K$ with $K \in \mathbb{N}$ the target set of all bit sequences of fixed size K, with $K < D$. It follows that messages $\boldsymbol{m} \in \mathcal{M}$ can only be mapped to 2^K different binary strings $\boldsymbol{b} \in \mathcal{B}$. The probability that a given $\boldsymbol{m}$ hashes to $\boldsymbol{b} = h(\boldsymbol{m})$ is $\frac{1}{K^2}$. Typically, conventional hashing is used for the realization of, e.g., hash tables, data integrity checks, error correction methods or database indexes. Such applications require a randomly uniform distribution of messages on hashes. Thus, a conventional hash function should therefore ideally assign hash values uniformly distributed with probability $\frac{1}{K^2}$ for all $\boldsymbol{m}$ and all $\boldsymbol{b}$. A related property in this context is the probability of a *collision*. A collision occurs when two messages $\boldsymbol{m}_1 \neq \boldsymbol{m}_2$ are projected onto the same hash value $h(\boldsymbol{m}_1) = h(\boldsymbol{m}_2)$.

For example, cryptographic hash functions are used in systems, where adversaries try to break such systems. Thus, different security requirements are defined for cryptograhpic hash functions [Wil, p. 349]. In particular, these hash functions are designed to be resistant against collisions. Applications that do not require the hash function to be resistant against adversaries, e.g., hash tables, caches or de-duplication, are usually implemented by using a non-cryptographic hash function that exhibits relaxed guarantees on the security properties in exchange for significant performance improvements. However, both cryptographic and non-cryptographic hash functions behave relatively similar with respect to the distribution of messages on hashes and the probability of a collision. In contrast, locality-sensitive hash functions behave more or less inversely as illustrated in Figure 2.7. A conventional hashing function h does not incorporate the similarity of messages into the calculation of the hash value. As a result, the messages are hashed evenly distributed into the slots of T_1. A locality-sensitive hashing function h_{LSH} on the other hand, hashes more similar messages into the same slot or neighbouring slots of T_2 as the hashes reflect the similarity of the input messages. In this regard, a hash table is introduced as an array T in which each element is a subset of messages that share a common hash value. Additionally, the position, i.e., slot of each subset in T is determined by that common hash value. For simplicity, we refer to a less mathematically sound notation within an algorithm description. Thus, the insert operation is denoted by $T[\boldsymbol{b}] = \boldsymbol{m}$, expressing that $\boldsymbol{m}$ is stored at slot $\boldsymbol{b} = h(\boldsymbol{m})$ within T. Conversely, the lookup operation is defined as $\mathcal{M}_{\boldsymbol{b}} = T[\boldsymbol{b}]$ for retrieving the subset of messages associated with the key $\boldsymbol{b}$.

As a locality-sensitive hashing function can be constructed as a general concept, specific families of functions can be derived. We refer to a *family* of hash functions $\mathcal{H} = \{h : \mathcal{M} \to \mathcal{B}\}$ as a collection of hash functions that have the same domain and range, share a basic structure and are only differentiated by constants. Three basic requirements are demanded for such a family of functions [LRU14, p. 99].

1. Close pairs of input should be hashed into the same bucket with higher probability than distant pairs.
2. Functions within a family need to be statistically independent from one another, such that the FPR and FNR can be improved by combining two or more functions.
3. Functions need to be efficient, i.e., be faster compared to an exhaustive search.

The first step is to define LSH generally. Applied on the cr-ANN, the first requirement states, with high probability, two points $\boldsymbol{p}, \boldsymbol{q}$ should hash to the same hash value if their distance is at most r, i.e., $d(\boldsymbol{p}, \boldsymbol{q}) \leq r$. And if their distance is at least cr, the points should hash to different hash values, i.e. $d(\boldsymbol{p}, \boldsymbol{q}) > cr$. Thus, A formal definition of a locality-sensitive hash function is given as follows [AI06].

Definition 5 (Locality-Sesitive Hash Function)**.** Given a target distance $r \in \mathbb{R}, r > 0$, an approximation factor $c \in \mathbb{R}, c > 1$ and probability thresholds $p_1, p_2 \in \mathbb{R}$, a family $\mathcal{H} = \{h : \mathcal{P} \to \mathcal{B}\}$ is called (r, cr, p_1, p_2)-sensitive if for any two points $\boldsymbol{p} \in \mathcal{P} \subset \mathcal{X}, \boldsymbol{q} \in \mathcal{X}$ and any hash function h chosen uniformly at random from $\mathcal{H}$ the following conditions are satisfied:

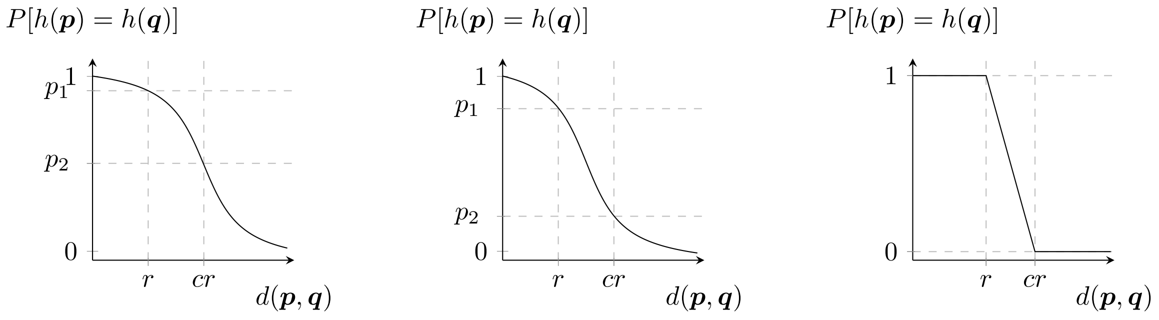
$$\begin{aligned} d(\boldsymbol{p}, \boldsymbol{q}) \leq r &\implies P[h(\boldsymbol{p}) = h(\boldsymbol{q})] \geq p_1, \\ d(\boldsymbol{p}, \boldsymbol{q}) \geq cr &\implies P[h(\boldsymbol{p}) = h(\boldsymbol{q})] \leq p_2. \end{aligned}$$

Ideally, the gap between p_1 and p_2 should be as big as possible as depicted in Figure 2.8c, which in fact represents an exact search. This is, as already discussed, no option due to its time and space requirements. Considering a single locality-sensitive function as shown in Figure 2.8a, where the probability gap between p_1 and p_2 is relatively close, the false negative rate would be relatively high. Increasing the gap would require to increase c and lead to a high number of false positives. Therefore, a single function would provide only a tradeoff. But it is possible to increase p_1 close to 1 and decrease p_2 close to $1/N$ while keeping r and cr fixed as shown in Figure 2.8b by introducing a process called *amplification*. Two different forms, namely the AND-construction and the OR-construction, can be applied. By applying a logical AND-construction on $\mathcal{H}$ the threshold p_2 is reduced. For that, K hash functions are sampled indepedently from $\mathcal{H}$ and each point $\boldsymbol{p}$ is hashed to a K-dimensional vector with a new constructed function $g \in \mathcal{H}^K$ as

$$g(p) = [h_1(\boldsymbol{p}), h_2(\boldsymbol{p}), \ldots, h_K(\boldsymbol{p})]. \tag{2.4}$$

Since all hash functions $\{h_1, \ldots, h_K\} \in \mathcal{H}$ are statistically independent the product rule applies and for any two points $\boldsymbol{p}$ and $\boldsymbol{q}$, a collision occurs if and only if $h_k(\boldsymbol{p}) = h_k(\boldsymbol{q})$ for all $k \in \{1, \ldots, K\}$. The probability gap is then defined as follows:

$$\begin{aligned} P[h_k(\boldsymbol{p}) = h_k(\boldsymbol{q})] \geq p_1 &\implies P[g(\boldsymbol{p}) = g(\boldsymbol{q})] \geq p_1^K \\ P[h_k(\boldsymbol{p}) = h_k(\boldsymbol{q})] \leq p_2 &\implies P[g(\boldsymbol{p}) = g(\boldsymbol{q})] \leq p_2^K. \end{aligned}$$



(a) Single LSH Function. (b) Amplified LSH Function. (c) Exact Search

Figure 2.8: The behaviour of a (r, cr, p_1, p_2)-sensitive function in (a) and (b) (adapted from [LRU14, p. 100]) approaching the ideal probability gap in (c) resembling the behaviour of an exact search.

Thus, by increasing K the threshold p_2 can be arbitrarily decreased and approaches zero. However, the AND-construction lowers both p_1 and p_2. In order to improve p_1, the OR-construction is introduced. For that, a number of $L \in \mathbb{N}$ functions $g_1, \ldots, g_L$ are constructed, where each function g_l, $l \in \{1, \ldots, L\}$ stems from a different family $\mathcal{H}_l$. Note, that the algorithm is successful, when the two points $\boldsymbol{p}, \boldsymbol{q}$ collide at least once for some g_l. For $d(\boldsymbol{p}, \boldsymbol{q}) \leq r$, the following probability for a collision follows as

$$\begin{aligned} P[\exists l \mid g_l(\boldsymbol{p}) = g_l(\boldsymbol{q})] &= 1 - P[\forall l, g_l(\boldsymbol{p}) \neq g_l(\boldsymbol{q})] \\ &= 1 - P[g_l(\boldsymbol{p}) \neq g_l(\boldsymbol{q})]^L \\ &\geq 1 - (1 - p_1^K)^L. \end{aligned}$$

As the AND-construction lowers both p_1 and p_2, similarly the OR-construction rises both thresholds. By choosing K and L judiciously, p_2 approaches zero while p_1 approaches one. Both constructions may be concatenated in any order to manipulate p_1 and p_2. Of course, the more constructions are used and the higher the values for the parameters K and L are picked, the more time is considered for the application of the function. The algorithm for solving the cr-ANN using LSH consists of two consecutive steps, namely the preprocessing (Algorithm 1) and the query (Algorithm 2).

```
Input: $N$ points $\boldsymbol{p}_n \in \mathcal{P}$ with $n \in \{1, \ldots, N\}, N \in \mathbb{N}$
Output: data structure $\mathcal{T} = \{T_1, \ldots, T_L\}$
$\mathcal{T} \leftarrow \emptyset$
construct hash functions $g_1, \ldots, g_L$ each of length $K$ (see Equation 2.4)
for each $g_l \in \{g_1, \ldots, g_L\}$ do
    $T_l \leftarrow$ new hash table
    for each $\boldsymbol{p}_n \in \mathcal{P}$ do
        $T_l[g_l(\boldsymbol{p}_n)] \leftarrow \boldsymbol{p}_n$
    add $T_l$ to $\mathcal{T}$
return $\mathcal{T}$
```

Algorithm 1: LSH Preprocessing

First, a set of points $\mathcal{P}$ is required for the construction of the data structure $\mathcal{T}$ as described in Algorithm 1. For each function g_l a corresponding hash table T_l is initialized that will store all data points $\boldsymbol{p}_n$. In other words, all data points are preprocessed and stored L times.The resulting data structure represents the database, which is searched through, when answering a query. Note, that a common collision handling method, such as seperate chaining for example, is applied if a certain slot is already occupied.

```
Input: a query point $\boldsymbol{q}$ and data structure $\mathcal{T} = \{T_1, \dots, T_L\}$
Output: a set $\mathcal{S}$ of nearest neighours of $\boldsymbol{q}$
$\mathcal{S} \leftarrow \emptyset$
for each $g_l \in \{g_1, \dots, g_L\}$ do
    $\mathcal{S}_l \leftarrow T_l[g_l(\boldsymbol{q})]$
    if $|\mathcal{S}_l| > 0$ then
        for each $\boldsymbol{p}_n \in \mathcal{S}_l$ do
            if $d(\boldsymbol{p}_n, \boldsymbol{q}) \leq cr$ then
                add $\boldsymbol{p}_n$ to $\mathcal{S}$
return $\mathcal{S}$
```

Algorithm 2: LSH Query

Answering a query point $\boldsymbol{q}$ is done by hashing it multiple times for each g_l as in the preprocessing step (see Algorithm 2). Each time, the set of points $\mathcal{P}$, stored in the correspoding hash table T_l at the slot $g_l(\boldsymbol{q})$, are retrieved. That is, identify all $\boldsymbol{p}_n$, such that $g_l(\boldsymbol{p}_n) = g_l(\boldsymbol{q})$. For each identified $\boldsymbol{p}_n$, a distance function evaluates if the distance $d(\boldsymbol{p}_n, \boldsymbol{q})$ is within the search perimeter cr. If positive, the point is added to the result set $\mathcal{S}$, which is returned at the end of the algorithm.

With regard to preprocessing, the consideration of space complexity of Algorithm 1 is interesting, whereas the runtime of Algorithm 2 is the most relevant when examining the execution of queries. Fortunately, lower bounds on both quantities have been proven in [MNP06], from which the optimization of the parameters K and L can be derived. The corresponding findings are summarized below.

Choose K such that $p_2^k = \frac{1}{N}$. Let ρ be the main parameter that determines the running time and space consumption of the algorithm. Assume that

$$p_1 = p_2^\rho,$$

for some $\rho < 1$ and choose $L \propto N^{-\rho} \ln N$. Then, by fixing a query point q it follows by linearity of expectation that for any l,

$$P[\exists \boldsymbol{p} \,|\, d(\boldsymbol{p}, \boldsymbol{q}) > cr, g_l(\boldsymbol{p}) = g_l(\boldsymbol{q})] = N p_2^K \leq 1.$$

By taking the sum over all l, in expectation there are L points $\boldsymbol{p} \in \mathcal{P}$ where $g_l(\boldsymbol{p}) = g_l(\boldsymbol{q})$ for some l, which implies an overhead of $O(L)$ in the query time. Conversely, if $d(\boldsymbol{p}, \boldsymbol{q}) \leq r$, then

$$\begin{aligned}
P[\exists l \mid g_l(\boldsymbol{p} = g_l(\boldsymbol{q})] &\geq 1 - (1 - p_1^K)^L \\
&= 1 - (1 - p_2^{\rho K})^L \\
&= 1 - (1 - n^{-\rho})^L \\
&\approx 1 - e^{LN^{-\rho}} \\
&= 1 - \frac{1}{N}.
\end{aligned}$$

Thus, for any point $\boldsymbol{q}$ whose distance is at most r, the algorithm finds $\boldsymbol{p}$ with a probability of at least $1 - \frac{1}{N}$. In expectation, the algorithm exhibits an overhead of $O(LD)$ for checking L points whose distance from $\boldsymbol{q}$ is at least cr.

The algorithm maintains a number of L hash tables. In each hash table N hash values are stored, where each hash value is a vector with K dimensions. Therefore, the space complexity of the algorithm is

$$O(LNK) = O(N^{1+\rho} \log \frac{N}{p_2}).$$

The query time for computing $g_l(\boldsymbol{q})$ for all $1 \leq l \leq L$ is $O(LK)$. For any candidate $\boldsymbol{p}$ the time to compute $d(\boldsymbol{p}, \boldsymbol{q})$ is $O(D)$. Let $|O|$ be the number of points whose distance from $\boldsymbol{q}$ is cr. In expectation, a number of L points are examined that are not returned by the algorithm, which results in a query time of $O(D(L + |O|))$ in expectation. Combined, the query time is

$$O(D(L + |O|)LK) = O(N^\rho(D + \log\frac{N}{p_2}) + |O|D).$$

By ignoring lower order terms, the algorithm exhibits a query time of $O(N^\rho)$ and a space complexity of $O(N^{1+\rho})$. Thus, the query time and space consumption improves significantly as ρ is decreased. In practice, the parameter ρ is based on the available resources.

2.2.3 Random Projection

This section presents a specific technique that is a locality-sensitive hashing function. The algorithm that bases on this technique is known as Gaussian Random Projection (GRP). This approach utilizes the cosine similarity of two real vectors in order to determine their similarity. In the course of this section, basic definitions for Cosine Distance and Cosine Similarity are introduced. Then, the individual calculation steps of the approach are specified. Finally, a visual proof verifies that gaussian random projection (GRP) is a form of LSH.

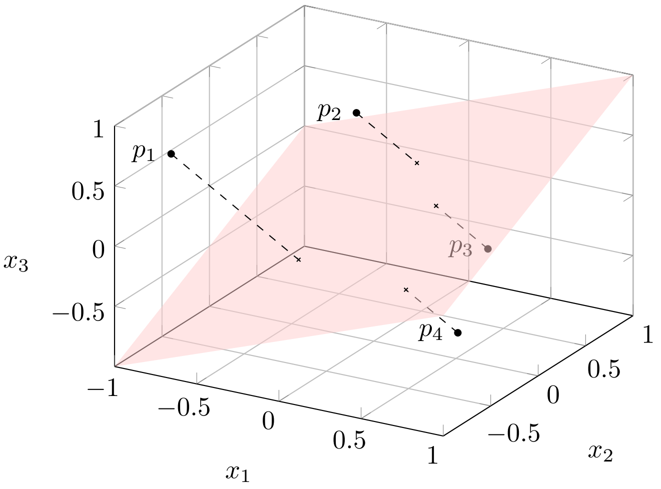


Figure 2.9: Illustration of a random hyperplane partitioning the space

The cosine distance can be applied in euclidean spaces and discrete versions of euclidean spaces [LRU14, p. 95]. For two real vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, the cosine distance between is equal to the the angle between $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, regardless of the dimensionality of the space. Note, that by applying the arc-cosine function, the result is in the range $[0, 180]$. The following definition formalizes what has been stated so far.

Definition 6 (Cosine Distance)**.** Given two vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, the cosine distance $\theta(\boldsymbol{p}_1, \boldsymbol{p}_2)$ is the dot product of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ divided by their euclidean distances from the origin (L_2-norm):

$$\theta(\boldsymbol{p}_1, \boldsymbol{p}_2) = \cos^{-1}\left(\frac{\boldsymbol{p}_1^\top \boldsymbol{p}_2}{||\boldsymbol{p}_1||\,||\boldsymbol{p}_2||}\right). \tag{2.5}$$

The angle θ can be normalized to the range $[0, 1]$ by dividing it by π. This way, the cosine similarity is simply given by the following definition.

Definition 7 (Cosine Similarity)**.** The cosine similarity is computed as

$$1 - \frac{\theta(\boldsymbol{p}_1, \boldsymbol{p}_2)}{\pi} \tag{2.6}$$

Introduced in [Cha02], the GRP is defined as follows. Given a point $\boldsymbol{p} \in \mathcal{P} \subset \mathbb{R}^D$ and a randomly selected hyperplane defined as $\boldsymbol{M} = (a_{ij}) \in \mathbb{R}^{D \times K}$ where $a \sim \mathcal{N}(0, I)$, a *gaussian random projection (GRP)* aims to (I) reduce the dimensionality from D to K dimensions and (II) provide a binary encoding by first projecting $\boldsymbol{p}$ onto $\boldsymbol{M}$ and subsequently applying the sign function to each element of the result, which can be formalized as

$$h(\boldsymbol{p}) = [h(\boldsymbol{p}, a_1), \ldots, h(\boldsymbol{p}, a_K)] \text{ with } h(\boldsymbol{p}, a) = sign(\boldsymbol{p}^\top a) \tag{2.7}$$

$$\text{with } sign(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases} \tag{2.8}$$

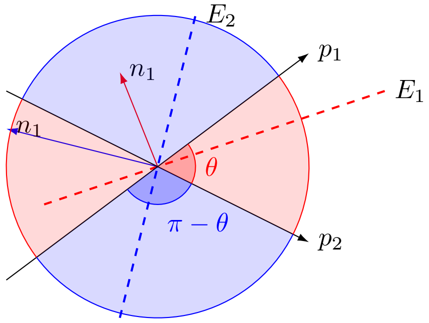


Figure 2.10: Visual Proof of claim in equation 2.9

An illistration of a random hyperplane that dissects the three-dimensional space and partitions the data space is given in Figure 2.9. The resulting digest is a binary vector $h(\boldsymbol{p}) = \boldsymbol{b} \in \mathcal{B}$ with $\mathcal{B} = \{0,1\}^K$ that is used as bucket index for storing $\boldsymbol{p}$ in a hash table. For any two messages $\boldsymbol{p}_1, \boldsymbol{p}_2$, the probability of being hashed to the same bucket increases with a decreasing distance, given as

$$P[h(\boldsymbol{p}_1) = h(\boldsymbol{p}_2)] = 1 - \frac{\theta(\boldsymbol{p}_1, \boldsymbol{p}_2)}{\pi}. \tag{2.9}$$

For a visual proof of the claim in Equation 2.9 consider Figure 2.10, where two vectors $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, regardless of their dimensionality, define a plane and an angle θ in this plane. Pick a hyperplane E_1 that intersects the plane that is spanned by $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ in a line (depicted as a red dashed line). Actually, a normal vector n_1 is randomly selected and E_1 is the set of points whose dot product with n_1 is zero. Since $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are on different sides of the hyperplane, their projections given by $\boldsymbol{p}_1^\top n_1$ and $\boldsymbol{p}_2^\top n_1$ will have different signs. Such a scenario, where two points have different signs, is interpreted as a notion of dissimilarity. Thus, the hyperplane acts as a clustering primitive that partitions the original input set into two disjunct sets.

The opposite scenario is illustrated by a random normal vector n_2 that is normal to the hyperplane E_2, which is represented by the blue dashed line. Both $\boldsymbol{p}_1^\top n_2$ and $\boldsymbol{p}_2^\top n_2$ will have the same sign. This in turn is interpreted as a notion of similarity, since both points are clustered into the same set. All angles between the intersection line of the random hyperplane and the plane spanned by $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ are equally likely. Thus, the probability that the hyperplane looks like the red line is θ/π and like the blue line $1 - \theta/\pi$.

Therefore, GRP is a $(r, cr, (1 - r/\pi), (1 - cr/\pi))$-sensitive family for any r and cr. As already explained in Section 2.2.2, such a family can be amplified as desired. The AND-construction is refers to the selection of a number of K different random hyperplanes r_k. According to the construction in Equation 2.7, all projections $sign(\boldsymbol{p}_n^\top n_k)$ for a single point are collected in a K-bit vector. Likewise, the OR-construction refers to the initialization of a number of L such AND-constructions. The algorithmic procedures for preprocessing and queries from Algorithm 1 and Algorithm 2 can be applied without change.

2.3 Gaussian Mixtures

GMMs refer to a density estimation technique the the machine learning context. With the help of density estimation techniques, large amounts of data can be compactly represented in a model. An estimate of important characteristics of an unobservable probability density function, such as mean or variance, is constructed based on the observed data. The density function is considered as the distribution according to which the observed data population is distributed. In the context of this work, Gaussian Mixture Models (GMMs) are used for the compact representation of attack data. Instead of exchanging the datasets directly in the CIDS, the data is instead represented by gmms, whereupon their parameters are shared. In this way, the data volume of the CIDS communication is significantly reduced. The GMM can be considered as a generative model, since new data can be generated by ancestral sampling. Thus, by reconstructing the GMMs from the shared parameters, the generation of synthetic data in the respective local infrastructures can be enabled. This data can in turn be used flexibly in the context of improving the respective intrusion detection.

For that purpose, importanct basics and concepts are introduced in this section. First, the Gaussian normal distribution is presented in Section 2.3.1 as the GMM is based on that parametric family. Then, in Section 2.3.2 mixture models in general and GMMs in particular are defined. Lastly, in Section 2.3.3 the GMM is interpreted in terms of a Latent Variable Model (LVM) and the Expectation Maximization Algorithm (EM Algorithm) is presented, which performs the computation of model parameters of latent variable models in an iterative scheme.

2.3.1 The Gaussian Distribution

The Gaussian normal distribution forms the basic building block for GMMs. Depending on the observed random variable, different types of normal distributions exist. In particular, the multivariate normal distribution is focused on when considering statistical flow features, which are typically continuous and high dimensional. In the following, different definitions for the normal distribution are introduced and important properties are presented that are essential for calculations during the learning phase of GMMs.

Let be a multivariate random variable $\boldsymbol{X} = (X_1, \ldots, X_D)^T$, where each element X_d with $d \in \{1, \ldots, D\}, D \in \mathbb{N}$ is a univariate random variable that follows a normal distribution, which is referred to as $X_d \sim \mathcal{N}(\mu, \sigma^2)$.

Definition 8 (Univariate Gaussian Distribution)**.** [DFO20, p. 175] A continuous, univariate random variable X is said to follow a normal distribution, if it exhibits a probability density function

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\bigg(- \frac{(x-\mu)^2}{2\sigma^2} \bigg),$$

with $x \in \mathbb{R}$ and where μ refers to the mean and σ^2 is the variance of the distribution. Then, if $\boldsymbol{X}$ follows a normal distribution, this is expressed as a multivariate Gaussian distribution, denoted by $\boldsymbol{X} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, which is fully described by its mean vector $\boldsymbol{\mu}$ and its covariance matrix $\boldsymbol{\Sigma}$.

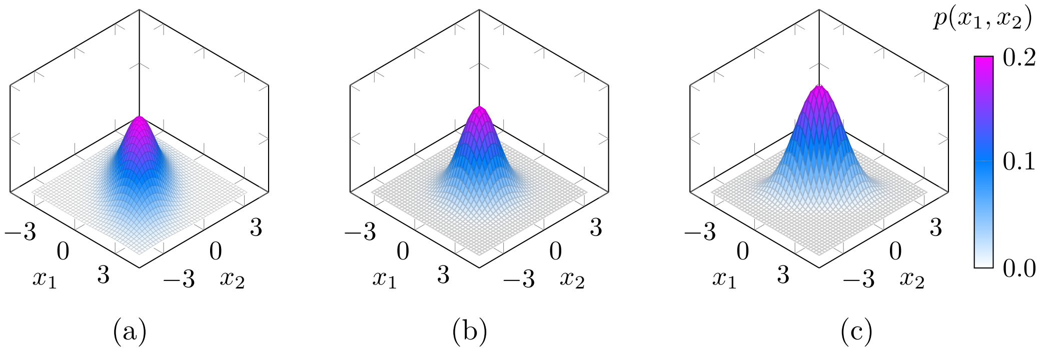


Figure 2.11: Bivariate gaussian distributions exhibit different shapes with a changing correlation value between the random variables x_1 and x_2: (a) negative, (b) zero and (c) positive correlation

Definition 9 (Multivariate Gaussian Distribution)**.** [DFO20, p. 175] A multivariate random variable $\boldsymbol{X}$ follows a normal distribution, if it is described by a probability density function

$$p(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}} exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right),$$

with $\boldsymbol{x} \in \mathbb{R}^D$. The mean vector specifies the estimate of the expected value of the distribution, with each of its components describing the mean μ of the corresponding dimension. The empirical covariance $\boldsymbol{\Sigma}$ models the estimate of the variance along each dimension as well as the correlation between the different dimensions. The diagonal elements of $\boldsymbol{\Sigma}$ describe the variance of the random variable corresponding to the respective dimension and the off-diagonal elements describe the covariance relationship between the respective random variables. An illustration of the influence of the described parameters on the location and shape of a multivariate distribution is given in Figure 2.11. Specifically, three bivariate normal distributions are considered, whose random variables each have different values with respect to their correlation, while the mean vector is the same for all distributions.

Having introduced the basic definition of Gaussian distributions, it is now examined how they can be manipulated to obtain information necessary for parameter learning of Gaussian mixture models. Two practical algebraic properties of normal distributions, namely closure under *conditioning* and *marginalization* are detailed for this purpose. Being closed under conditioning and marginalization in this case means that when one or more components of a Gaussian distribution is marginalized out or conditioned on, that the resulting distribution is still Gaussian. This not only distinguishes Gaussian distributions from other distributions, but also makes them easier to handle mathematically. Since both marginalization and conditioning act on subsets of the original distribution, the following notation is introduced first. Considering a multivariate Gaussian random variable $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with $\boldsymbol{X}_M \in \mathbf{R}^M$, where $M < N$, and $\boldsymbol{X}_N \in \mathbf{R}^N$, where $N = D - M$, we partition $\boldsymbol{X}$ according to

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_M \\ \boldsymbol{X}_N \end{bmatrix}.$$

In general, $\boldsymbol{X}_M$ is chosen to be the first M elements of $\boldsymbol{X}$ and $\boldsymbol{X}_N$ the rest. Accordingly, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are partitioned as

$$\begin{bmatrix} \boldsymbol{X}_M \\ \boldsymbol{X}_N \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_M \\ \boldsymbol{\mu}_N \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{MM} & \boldsymbol{\Sigma}_{MN} \\ \boldsymbol{\Sigma}_{NM} & \boldsymbol{\Sigma}_{NN} \end{bmatrix} \right).$$

With this form, it is now possible to express the extraction of partial information from multivariate probability distributions by means of marginalization.

Definition 10 (Marginal of a Gaussian Distribution)**.** [DFO20, p. 177] Given $\boldsymbol{X} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with partitions $\boldsymbol{X}_M, \boldsymbol{X}_N$, the distributions of $\boldsymbol{X}_M$ and $\boldsymbol{X}_N$ are called marginals and their corresponding probability density function can be obtained by

$$p(\boldsymbol{x}_M) = \int p(\boldsymbol{x}_M, \boldsymbol{x}_N) d\boldsymbol{x}_N.$$

Furthermore, each partition $\boldsymbol{X}_M, \boldsymbol{X}_N$ only depend on its corresponding entries in $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, which leads to the following theorem.

Theorem 11. [DFO20, p. 177] The marginal distribution of a Gaussian distribution is also a Gaussian and determined by

$$\boldsymbol{X}_M \sim \mathcal{N}(\boldsymbol{\mu}_M, \boldsymbol{\Sigma}_{MM}).$$

The conditional Gaussian is typically utilized in the context of posterior distributions, such as in the process of density estimation of GMMs, and is defined as follows.

Definition 12 (Conditional of a Gaussian Distribution)**.** [DFO20, p. 177] Given $\boldsymbol{X} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with partitions $\boldsymbol{X}_M, \boldsymbol{X}_N$, the conditional distribution is defined as

$$\begin{aligned} p(\boldsymbol{x}_M|\boldsymbol{x}_N) &= \mathcal{N}(\boldsymbol{\mu}_{M|N}, \boldsymbol{\Sigma}_{M|N}), \\ \boldsymbol{\mu}_{M|N} &= \boldsymbol{\mu}_M + \boldsymbol{\Sigma}_{MN} + \boldsymbol{\Sigma}_{NN}^{-1}(\boldsymbol{x}_N - \boldsymbol{\mu}_N), \\ \boldsymbol{\Sigma}_{M|N} &= \boldsymbol{\Sigma}_{MM} - \boldsymbol{\Sigma}_{MN}\boldsymbol{\Sigma}_{NN}^{-1}\boldsymbol{\Sigma}_{NM}. \end{aligned}$$

Theorem 13. [DFO20, p. 177] The conditional distribution of a Gaussian distribution is also a Gaussian and defined by

$$\boldsymbol{X}_{M|N} \sim \mathcal{N}(\boldsymbol{\mu}_{M|N}, \boldsymbol{\Sigma}_{M|N}).$$

A line of arguments proving the stated theorems can be found in section 2.3 in [Bis06]. An example for marginal and conditional Gaussians can be found in Figure 5.1. The functions of $p(x_M)$ and $p(X_N)$ are plotted on the side grids. The conditional cuts through the plot of the joint distribution $p(x_M, x_N)$.

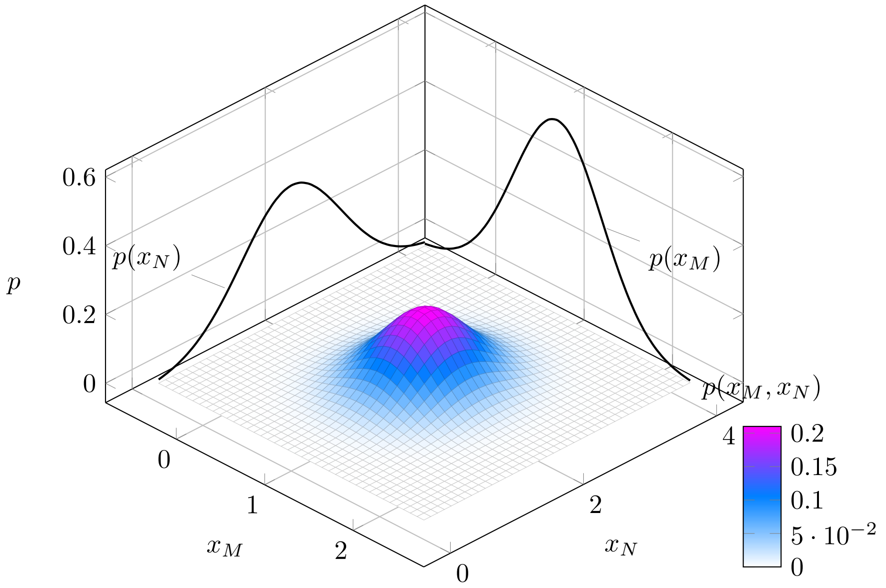


Figure 2.12: Marginals $p(x_M)$, $p(x_N)$ and Conditional Gaussian $p(x_{M|N})$

2.3.2 The Gaussian Mixture Model

Using statistical estimation techniques, it is possible to estimate an unobservable underlying probability density function from observed data. This allows data to be compactly represented with a density from a parametric family, such as a Gaussian distribution. However, all conventional parametric distributions are limited in their modeling capabilities when confronted with real data. For example, considering data that follows a multimodal distribution, i.e., have more than one center, a density estimate using a simple Gaussian distribution is not sufficient to effectively represent data distribution.

The idea is to represent a multimodal distribution by constructing a linear combination of multiple simple distributions, each of which representing a unimodal sub-population of the data, which is formalized under the term *mixture model* [Bis06, p.111].

Definition 14 (Mixture Model)**.** [DFO20, p. 315] A mixture model is a linear combination of K parametric distributions p_k, each weighted by a mixture weight π_k, with the following form

$$p(x) = \sum_{k=1}^{K} \pi_k p_k(x),$$
$$0 \leq \pi_k \leq 1, \sum_{k=1}^{K} \pi_k = 1.$$

A distribution p_k within this model is called mixture component and the sum of the mixture weights equals to 1, such that the probability density of the mixture components equals to 1 as well.

Mixture Models can use any arbitrary parametric distribution as component density, but the most common mixture model is the Gaussian mixture model, using Gaussians as components [HTF09, p. 214]. First, GMMs utilize the practical mathematical properties of Gaussians, introduced earlier in Section 2.3.1.

Furthermore, the quality of the model, in terms of its ability to estimate real data distributions, is theoretically supported by the Central Limit Theorem (see Definition 15), which states, that most data distributions converge to a normal distribution on average as the number of data points increases [JB03, p.222].

Theorem 15 (Central Limit Theorem)**.** [MR10, p.241] Consider N i.i.d. random variables X_i with $\mathbb{E}[X_i] = \mu$ and $\mathbb{V}[X_i] = \sigma^2$ and let $S_N = \sum_{i=1}^{N} X_i$. It can be shown that, as N increases, that

$$p(S_N) \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{N}\right).$$

By further specifying the general definition of mixture models (see Definition 14), we obtain the definition of a GMM (see Definition 16). Instead of using an arbitrary parametric distribution as the component density, the Gaussian distribtion is utilized. Furthermore, the set of parameters $\boldsymbol{\theta}$ of the GMM is introduced.

Definition 16 (Gaussian Mixture Model)**.** [DFO20, p. 315] A Gaussian Mixture Model is a combination of a finite number of K Gaussian distributions $\mathcal{N}(\boldsymbol{x}|\theta_k)$ which is fully described by a probability density function p and its parameter set $\boldsymbol{\theta}$ as

$$\begin{aligned}
&p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x} \,|\, \theta), \\
&0 \leq \pi_k \leq 1, \sum_{k=1}^{K} \pi_k = 1, \\
&\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k \,|\, k = 1, \dots, K\}.
\end{aligned} \tag{2.10}$$

2.3.3 The Expectation Maximization Algorithm

So far, with the definition and the example, an intuitive explanation of the GMM has been given. Before it is explained how to estimate the parameters of the model, it is necessary to look at the subject from the perspective of probabilistic modeling. Probabilistic models, in general, utilize the mathematics of probability theory in order to express all forms of uncertainty and noise that is associated with the learning task. If a Bayesian interpretation is thereby applied to the probabilities, then this is commonly referred to as Bayesian Inference. This allows, for example, the estimation of the parameters of a probability distribution or a statistical model utilizing Bayes' Theorem [DFO20, p.245]. In this case, discrete latent variables must be introduced into the construction, which allows a probabilistic model to be formed by defining a joint distribution over observed variables and latent variables [Bis06, p. 432].

The term latent variable usually refers to a variable that cannot be observed directly, but can be inferred from other observable variables using mathematical models [BMH03]. Models involving such latent variables, called LVM, are usually harder to fit than models without latent variables, but often have fewer parameters due to their natural implication of a bottleneck, resulting in a compressed representation of the data [Mur12, p. 337].

For simplicity, first a single data point $\boldsymbol{x}$ is considered, which is later expanded to a set of data points $\mathcal{X} := \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}, \mathcal{X} \subset \mathbb{R}^D$. It is further assumed a GMM with K components. Thus, a K-dimensional binary random variable $\boldsymbol{z}$ is introduced.

Note that $\boldsymbol{z} = [z_1, \ldots, z_K]^\top$ indicates whether the kth mixture component is responsible for generating the data point $\boldsymbol{x}$ and that a data point can only be generated by one mixture component. Hence, only a particular element z_k is equal to one and all other elements are equal to zero, such that

$$z_k \in \{0, 1\}, \quad \sum_{k=1}^{K} z_k = 1. \tag{2.11}$$

For the construction of a probabilistic model, it is necessary to specify the joint distribution of the observed data $\boldsymbol{x}$ and the latent variable $\boldsymbol{z}$. Therefore, the factors of the joint distribution are defined as

$$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}). \tag{2.12}$$

The responsibility of mixture component k generating a data point $\boldsymbol{x}$ can be expressed as the conditional distribution of $\boldsymbol{x}$ given a specific assignment for $\boldsymbol{z}$, which is a Gaussian

$$p(\boldsymbol{x}|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad \Rightarrow \quad p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}. \tag{2.13}$$

In practice, there exists no knowledge about the value assignment of the latent variables. Therefore, a prior is set to $\boldsymbol{z}$, which is defined as

$$p(\boldsymbol{z}) = \boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]^\top, \quad \sum_{k=1}^{K} \pi_k = 1. \tag{2.14}$$

Now, there is a way to describe the *probability* that the kth mixture component generated the data point $\boldsymbol{x}$ as

$$p(z_k = 1) = \pi_k. \tag{2.15}$$

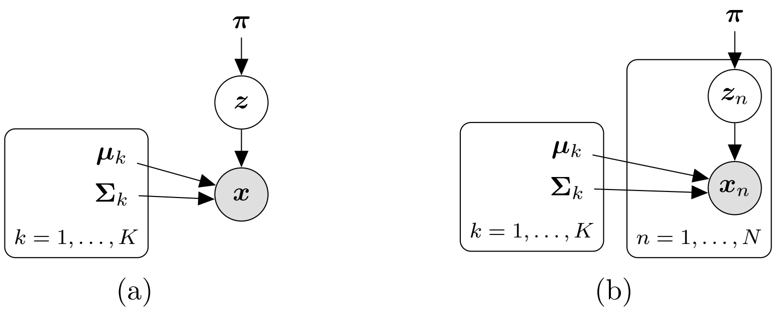


Figure 2.13: Both graphs represent a GMM using plate notation. While the left graph (a) includes a single data point $\boldsymbol{x}$, the graph on the right side (b) extends the model to a full dataset with N i.i.d. data points $\{\boldsymbol{x}_n\}$ and corresponding latent variables $\{\boldsymbol{z}_n\}$. Note, that the directed edge from the latent variable to the observed data reflects the joint distribution from (2.12).

Then the conditional distribution $p(\boldsymbol{x}|\boldsymbol{z})$ from (2.13) and the prior distribution $p(\boldsymbol{z})$ from (2.14) are plugged into the joint distribution $p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$ from (2.12) and the marginal distribution $p(\boldsymbol{x})$ is obtained by summing the joint distribution over all possible states of $\boldsymbol{z}$, which results in

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2.16}$$

It can be summarized that the GMM has been extended by including the discrete latent variable $\boldsymbol{z}$ and that the result is still consistent with the previous definition, since the marginal distribution $p(\boldsymbol{x})$ is a Gaussian mixture of the form (2.10). This enables the joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$ to be used in place of the marginal distribution $p(\boldsymbol{x})$, greatly simplifying parameter estimation by applying the EM Algorithm.

So far, it was assumed that the dataset consists of a single data point $\boldsymbol{x}$. Now, the dataset is extended to N data points. From that, it follows that every data point $\boldsymbol{x}_n$ possesses its own latent variable $\boldsymbol{z}_n$. All latent variables can be summarized by $\mathcal{Z} \subset \mathbb{R}^K$. This extension to a full dataset is also illustrated comparatively in Figure 2.13. In accordance to that, the corresponding posterior probability after observing $\boldsymbol{x}$, referred to as r_{nk}, is obtained using Bayes' Theorem as

$$\begin{aligned} r_{nk} = p(z_{nk} = 1|\boldsymbol{x}_n) &= \frac{p(z_{nk} = 1)p(\boldsymbol{x}|z_k = 1)}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x}_n)|z_{nk} = 1)p(z_{nk} = 1)}{\sum_{j=1}^{K} \pi_j p(\boldsymbol{x}_n|z_{nj} = 1)p(z_{nj} = 1)} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n)|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n)|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \end{aligned} \tag{2.17}$$

The posterior r_{nk} can be interpreted, from the perspective of a generative model, as the proportion with which a component is involved in the generation of the point $\boldsymbol{x}_n$.

The central question is how to fit the set of unknown parameters $\boldsymbol{\theta}$ to a given set of data $\mathcal{X}$ and therefore finding a good approximation for the unknown distribution $p(x)$. For estimation problems based on i.i.d. data points, the Maximum Likelihood Estimation (MLE) is an applicable method [DFO20, p. 317]. By doing this, the likelihood of each data point $\boldsymbol{x} \in \mathcal{X}$ given a specific parametrization $\boldsymbol{\theta}$ is maximized. To do so, the likelihood function, and the log-likelihood function respectively, must first be constructed, where each individual likelihood term is a Gaussian mixture density as

$$p(\mathcal{X}|\theta) = \prod_{n=1}^{N} p(\boldsymbol{x}_n|\boldsymbol{\theta}), \quad p(\boldsymbol{x}_n|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

$$\log p(\mathcal{X}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}_n|\boldsymbol{\theta}) = \underbrace{\sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma})}_{=:\mathcal{L}}.$$

The common solution scheme that would follow would be to calculate the gradient $\partial\mathcal{L}/\partial\boldsymbol{\theta}$, set it to zero and solve for $\boldsymbol{\theta}$. Unfortunately, there is no closed form solution for a MLE in this form, because the summation over the K components prevents the logarithm from being applied to the Gaussian densities within [Bis06, p. 435]. That means that each of the partial derivatives of the model parameters depends on all K parameters of the GMM through r_{nk}, which prohibits a closed form solution.

However, a solution can be found using the EM Algorithm. The key idea here is to update one model parameter at a time while keeping the others fixed. For that, the ML estimators for the individual parameters of the GMM are required first, which will be applied in the EM Algorithm in the following step. The following necessary conditions are established:

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\boldsymbol{\mu}_k} = \mathbf{0}^T \quad &\iff \quad \sum_{n=1}^{N}\frac{\partial \text{log}p(\boldsymbol{x}_n|\theta)}{\partial\boldsymbol{\mu}_k} = \mathbf{0}^\top \\
\frac{\partial\mathcal{L}}{\partial\boldsymbol{\Sigma}_k} = \mathbf{0} \quad &\iff \quad \sum_{n=1}^{N}\frac{\partial \text{log}p(\boldsymbol{x}_n|\theta)}{\partial\boldsymbol{\Sigma}_k} = \mathbf{0} \\
\frac{\partial\mathcal{L}}{\partial\pi_k} = 0 \quad &\iff \quad \sum_{n=1}^{N}\frac{\partial \text{log}p(\boldsymbol{x}_n|\theta)}{\partial\pi_k} = 0.
\end{aligned}$$

In the following only the partial derivatives of the model parameters that will be used in the EM Algorithm are presented. A detailed calculation can be found in [DFO20, pp.319].

$$\begin{aligned}
\boldsymbol{\mu}_k' &= \frac{\sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}} \\
\boldsymbol{\Sigma}_k' &= \frac{1}{N_k}\sum_{n=1}^{N} r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \qquad (2.18) \\
\pi_k' &= \frac{N_k}{N}
\end{aligned}$$

The EM Algorithm is an iterative procedure that starts with an initial estimate of the parameters, which are updated incrementally until convergence is detected. The initial values for the parameters can either be chosen randomly or obtained with an heuristic method, e.g. by using k-means to cluster the data first and subsequently defining the weights based on the resulting k-means memberships [DFO20, pp. 325].

Each iteration t of the EM Algorithm consists mainly of two steps, the Expectation step and the Maximization step. A third step is subsequently needed for the calculation of the convergence criterion. First, in the expectation step, the posterior distribution r_{nk} from (2.17) is calculated for all data points $\boldsymbol{x}_n$ with the current set of parameters $\boldsymbol{\theta}^{(t)}$. Then, the posterior from the previous step is used in the maximization step for computing the new parameter values with the Maximum-Likelihood estimators defined in (2.18). At the end of each iteration, the log likelihood for the new set of parameters $\boldsymbol{\theta}^{(t+1)}$ and its change from the log likelihood from the previous iteration is computed. Then, the algorithm can test for convergence by comparing $\Delta\mathcal{L}$ with the convergence threshold ϵ and the algorithm stops if no significant change occur anymore.

```
Initialize:
    $t = 0$
    $\boldsymbol{\theta}^{(t)} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k \mid k = 1, \dots, K\}$
    $\mathcal{L}^{(t)} = \log p(\mathcal{X} \mid \boldsymbol{\theta}^{(t)})$
while $\Delta\mathcal{L} > \epsilon$ do
    1. Expectation-Step:
        $r_{nk} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$ with $\boldsymbol{\theta}^{(t)}$
    2. Maximization-Step:
        $\pi_k^{(t+1)} = \frac{N_k}{N}$
        $\boldsymbol{\mu}^{(t+1)} = \frac{\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}}$
        $\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top$
    Convergence Terms:
        $\mathcal{L}^{(t+1)} = \log p(\mathcal{X} \mid \boldsymbol{\theta}^{(t+1)})$
        $\Delta\mathcal{L} = |\mathcal{L}^{(t+1)} - \mathcal{L}^{(t)}|$
        $t = t + 1$
```

Algorithm 3: EM Algorithm for GMM

Figure 2.14 illustrates the EM Algorithm for fitting a GMM with three components on a two-dimensional synthetic dataset with three classes. Each component of the GMM is represented by an coloured ellipse, that is formed by its mean and covariance matrix. Every single data point $\boldsymbol{x}_n$ is associated with an probability vector, that describes each component's responsibility for creating $\boldsymbol{x}_n$. A data point is assigned to the component that exhibits the highest responsibility value. This is visualized by coloring each data point according to their assignment to a component. In this example, the three component parameters are chosen randomly and no posterior r_{nk} is computed in the beginning ($t = 0$) and thus no data point is colored. In this state, the true class labels can bee seen in different grayscales.

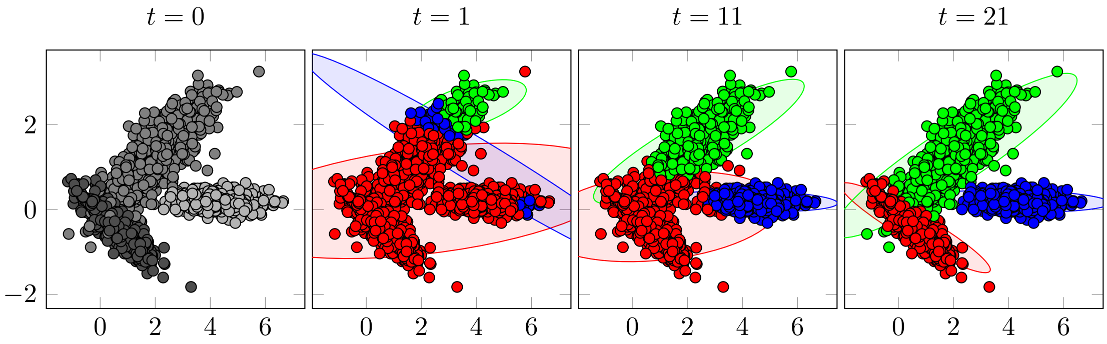


Figure 2.14: Illustration of the EM algorithm for fitting a GMM with three components on a two-dimensional dataset.

With each iteration, the parameters approach the local optimum, which they reach after iteration 21 ($t = 21$), and the algorithm converges. In this toy example, the three classes can be seperated well by three components. Note that multiple components may describe a single class as well. This raises the question of how many components should be ideally chosen for fitting a GMM, when the number of classes are not known beforehand as in this example?

A common method is to select the model with the highest probability given the data $\mathcal{X}$. Recall that the likelihood $\mathcal{L}$ answers the question of what is the probability that $\mathcal{X}$ is explained by the model. Theoretically, in the case of a GMM, one could just arbitrarily increase the number of components K until $K = N$ in order to obtain the maximum likelihood. In practice, this would not only lead to a bad runtime of the algorithm, but also overfit the model. Thus, the maximum likelihood of the model $\mathcal{L}$ has to be balanced against the number of model parameters. The bayesian information criterion (BIC) is considered as a standard criterion for GMM model selection because of its theoretical consistency in choosing the number of components [Ker00]. Let $|\boldsymbol{\theta}|$ be the number of model parameters in general, then the BIC is defined as

$$\mathrm{BIC} = |\boldsymbol{\theta}| \ln(N) - 2\ln(\mathcal{L}), \tag{2.19}$$

which derives from the findings in [Sch78]. The BIC balances the number of model parameters $|\boldsymbol{\theta}|$ and number of data points N against the likelihood function $\mathcal{L}$. In the model selection, the optimal number of model parameters $|\boldsymbol{\theta}|$ minimizes the BIC, such that it provides a principled way of selecting between multiple different models. More complex models almost always fit the data better, resulting in a lower value for the term $-2\ln(\mathcal{L})$. The BIC penalizes extra parameters by introducing the term $|\boldsymbol{\theta}| \ln(N)$. A model selection utilizing the BIC is described in Algorithm 4, with $\boldsymbol{\theta}$ being the set of parameter settings for GMM.

```
**Input:** Dataset $\mathcal{X}$
**Output:** Gaussian Mixture Model GMM
$L_{\mathrm{GMM}} \leftarrow$ new List
**for each** $\theta$ in $\boldsymbol{\theta}$ **do**
    $\mathrm{GMM}_k \leftarrow$ fitGaussianMixture($\mathcal{X}, \theta$)
    $\mathrm{BIC}_k \leftarrow$ computeBIC(GMM$|\mathcal{X}$)
    append ($\mathrm{GMM}_k, \mathrm{BIC}_k$) to $L_{\mathrm{GMM}}$
sort $L_{\mathrm{GMM}}$ ascending by $\mathrm{BIC}_k$
GMM $\leftarrow L_{\mathrm{GMM}}[0]$                    ▶ get head of the list
**return** GMM
```

Algorithm 4: GMM Selection with BIC

To be more precisely, the parameters of K multivariate normal distributions are to learn. With $D = \dim(\mathcal{X})$, these are D values in the mean vector. Since a covariance matrix is symmetric, only $D(D + 1)/2$ entries in a full covariance matrix have to be computed. Additionaly, K mixture weights have to be determined, leading to $Kd + K(d(d+1)/2) + K$ parameters. Thus, the BIC for a dataset $\mathcal{X}$ with N datapoints of dimensionality D and a GMM as defined above is stated as

$$\mathrm{BIC}(\mathrm{GMM}|\mathcal{X}) = (KD + K(D(D+1)/2) + K)\ln(N) - 2\ln(\mathcal{L}). \tag{2.20}$$

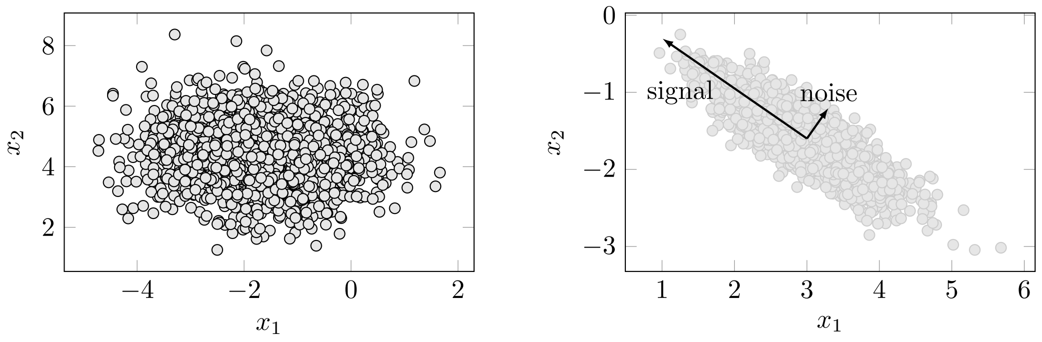
2.4 Principal Component Analysis

This chapter focuses on dimensionality reduction of data in the context of Principal Component Analysis (PCA). First, the application of dimensionality reduction in general is motivated by various examples. Then, the core idea based on the separation of signal and noise in data is presented. After this, a concrete statement of the problem is introduced. In addition, key model assumptions and a solution approach is presented and followed by a mathematical derivation of the eigencomposition for solving PCA.

2.4.1 Motivation

Several difficulties arise when analyzing high-dimensional data [DFO20, p.286]. For example, it is often not easy to interpret which of the many factors have the most significant influence on the data distribution. In addition, data visualizations are often limited to a few dimensions. And besides the impact on data analysis, there are also influences on data inference. Some machine learning algorithms cannot effectively reason about the underlying structure of the data in high-dimensional space, which can result into a model overfitting. Furthermore, compression of data, in terms of dimensionality reduction, can lead to advantages in processing and persistence of data. Data storage can often come at a high cost and some algorithms do not scale well with increasing numbers of dimensions. For example, when using the EM Algorithm (see Section 2.3.3) an increase of the dimensionality of the data is related to an increase in the probability of the occurrence of (nearly) singular matrices, which can result in a numerical breakdown of the model [Bis06, p.434]. For these reasons, dimensionality reduction is often desired within a data preprocessing strategy.

In general, dimensionality reduction methods make use of the following idea. Measuring data is error prone and associated with both incompleteness and redundancy. This trend increases with the number of dimensions. Thus, many dimensions in multivariate data can be explained by a combination of other dimensions. This underlying structure can be exploited by partitioning the full space into subspaces of signal and noise [Sco15, p. 217]. Figure 2.15 shows two synthetic datasets each with two features. The plot on the right displays a negative correlation and the direction that carries the largest portion of the signal can be located visually.



(a) uncorrelated data

(b) correlated data

Figure 2.15: Two synthetic datasets illustrating the difference in signal and noise

In this two-dimensional example, the direction that is perpendicular to that is the direction that carries the least amount of signal and is referred to as noise. The goal is to discard a significant amount of noise in the data and work with a more compact representation, with losing as little signal as possible.

2.4.2 Problem Description

All data points $\boldsymbol{x}_n$ can be represented as a linear combination of the orthonormal basis. The central question associated with PCA is whether there is another basis, which is a *linear* combination of the original basis, that best represents the data. From this perspective, PCA can be considered a change-of-basis problem. Recalling the example from Figure 2.15, a better representation of the data is related to a lower amount of noise and redundancy in the data. Further, by assuming linearity, the problem is simplified by restricting the set of potential bases and enabling an efficient solution with matrix decomposition techniques [Shl14]. Note that it is not always the case that the correlation among variables is not of interest. But in the case of PCA, correlation represents redundancy, which is desired to be minimized.

To derive a solution approach, some assumptions are made beforehand, which are proven in the following section. It is assumed that data containing more redundancy generally have lower variance and vice versa. Therefore, the dynamics of interest exist in the directions of highest variance. It is further assumed that the canonical basis in which the data was captured is a representation that has inherent redundancies and noise. For this reason, a suitable basis is searched that represents a linear transformation of the canonical basis, such that the base aligns with the axes of maximum variance of the data. By projecting the data onto the axis of maximum variance, which form a lower-dimensional subspace, data redundancy is eliminated.

Finding the directions of maximum variance, which are referred to as principal components, is the central operation of PCA. One feasible option is to diagonalize the covariance matrix of the data. Recalling that each covariance matrix is a square symmetric matrix whose off-diagonal elements represent the covariances, i.e., reflect the redundancies, a diagonalization of the covariance matrix eliminates the magnitude of the covariance, resulting in a matrix that only contains the variance terms. Finally, each successive dimension in the diagonalized covariance matrix should be rank-ordered according to the values of the variance in order to quantify their importance. It can be shown that the principal components are the eigenvectors of the covariance matrix of the data. A frequently used method for solving PCA is therefore the eigendecomposition, which is mathematically derived in the following in connection with the assumptions made so far.

2.4.3 Formal Description

Consider a dataset $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^D$ with mean $\mathbf{0} = [0_1, \ldots, 0_N]^\top$ and a covariance matrix

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top .$$

In terms of dimensionality reduction, a compressed representation of the dataset $\mathcal{X}$, where each datapoint is defined as

$$\boldsymbol{z_n} = \boldsymbol{B}^T \boldsymbol{x}_n \in \mathbb{R}^M$$

is searched for, which retains as much signal as possible [Hot33]. This is done by finding a linear transformation

$$\boldsymbol{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M] \in \mathbb{R}^{D \times M}$$

that projects $\mathcal{X}$ onto the subspace that is spanned by the columns of $\boldsymbol{B}$. Thus, $\boldsymbol{B}$ is an orthonormal basis. In order to preserve the most variance of the data, the basis vectors of $\boldsymbol{B}$ must point into the direction of maximal variance in the data, which is found by computing the eigenvectors of the covariance matrix $\boldsymbol{\Sigma}$ of $\mathcal{X}$. The projection of the data onto a lower-dimensional subspace is equal to the eigenvalue that is associated with the basis vector that spans this subspace.

Therefore, maximizing the variance of the low-dimensional representation requires to choose the basis vector that is associated with the largest eigenvalue of the data covariance matrix. This eigenvector is called the first principal component. All other directions in $\mathbb{R}^M$, which both maximize the variance and are orthogonal to all the other directions are the remaining $M - 1$ principal components.

As stated before, the projection matrix $\boldsymbol{B}$ is obtained by diagonalizing the covariance matrix $\boldsymbol{C}$, which can be done with via eigendecomposition. In the following, it is shown how the eigendecomposition is related to the diagonalization of a matrix and that an eigendecomposition of a covariance matrix is guaranteed to exist.

Theorem 17 (Matrix Diagonalization)**.** [DFO20, p.98] A matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is diagonalizable if it is similar to a diagonal matrix, i.e., if there exists an invertible matrix $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ such that $\boldsymbol{D} = \boldsymbol{P}^{-1}\boldsymbol{AP}$.

Theorem 18 (Eigendecomposition)**.** [DFO20, p.99] A square matix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ can be factorized into

$$\boldsymbol{A} = \boldsymbol{PDP}^{-1},$$

where $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{D}$ is a diagonal matrix whose entries are the eigenvalues of $\boldsymbol{A}$, if and only if the eigenvectors of $\boldsymbol{A}$ form a basis of $\mathbb{R}^n$.

It follows, that only non defective matrices can be diagonalized. In other words, a square matrix that does not have a complete basis of eigenvectors is not diagonalizable. However, the following theorem confirms the existence of such a basis for symmetric matrices.

Theorem 19 (Spectral Theorem)**.** [DFO20, p.94] If $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is symmetric, there exists an orthonormal basis of the corresponding vector space consisting of eigenvectors of $\boldsymbol{A}$, and each eigenvalue is real.

From the fact that every covariance matrix is symmetric, the following can be concluded.

Theorem 20 (Diagonalization of a Symmetric Matrix)**.** A symmetric matrix $\boldsymbol{S} \in \mathbb{R}^{n \times n}$ can always be diagonalized.

Therefore, it is assured that an eigendecomposition of a symmetric matrix $\boldsymbol{A}$ exists (with real values), and that an ONB of eigenvectors can be found, so that $\boldsymbol{A} = \boldsymbol{PDP}^{-1}$, where $\boldsymbol{D}$ is diagonal and the columns of $\boldsymbol{P}$ contain the eigenvectors. Rewriting the equation from Theorem 18 as follows reveals the eigenvalue equations:

$$\boldsymbol{A}[p_1, \ldots, p_N] = [p_1, \ldots, p_N] \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_N \end{bmatrix} \iff \begin{array}{rcl} \boldsymbol{A}p_1 & = & \lambda_1 p_1 \\ \boldsymbol{A}p_2 & = & \lambda_2 p_2 \\ \boldsymbol{A}p_3 & = & \lambda_3 p_3 \\ & \cdots & \\ \boldsymbol{A}p_N & = & \lambda_N p_N \end{array}$$

Next, it is shown how the eigenvalue equation shown above is related to the maximization of the variance of the data. For this purpose, the single vector $\boldsymbol{b}_1 \in \mathbb{R}^D$ is considered only, which is the first column of the matrix $\boldsymbol{B}$ and therefore the first of M orthonormal basis vectors. By exploiting the i.i.d. assumption of the data, the first component of $\boldsymbol{z}_n \in \mathbb{R}^M$ of a single data point $\boldsymbol{x}_n \in \mathbb{R}^D$ is given by

$$z_{1n} = \boldsymbol{b}_1^\top \boldsymbol{x}_n. \tag{2.21}$$

Thus, z_{1n} is the coordinate of the orthogonal projection of $\boldsymbol{x}_n$ onto the one-dimensional subspace spanned by $\boldsymbol{b}_1$. Then, the variance of z_{1n} of $\boldsymbol{z} \in \mathbb{R}^M$, that is maximized by $\boldsymbol{b}_1$, is defined as

$$\mathbb{V}_1 := \mathbb{V}[z_1] = \frac{1}{N} \sum_{n=1}^{N} z_{1n}^2. \tag{2.22}$$

The relation between the variance and the factorized matrix in Theorem 18 becomes evident by substituting equation (2.21) into equation (2.22) as

$$\begin{aligned} \mathbb{V}_1 &= \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{b}_1^\top \boldsymbol{x}_n)^2 \\ &= \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{b}_1^\top \boldsymbol{x}_n \boldsymbol{x}_n^\top \boldsymbol{b}_1 \\ &= \boldsymbol{b}_1^\top (\frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top) \boldsymbol{b}_1 \\ &= \boldsymbol{b}_1^\top \boldsymbol{\Sigma} \boldsymbol{b}_1. \end{aligned} \tag{2.23}$$

Maximizing the variance defined in (2.23) results in finding vector $\boldsymbol{b}_1$. But arbitrarily increasing the magnitude of $\boldsymbol{b}_1$ increases $\mathbb{V}_1$, which is why restricting all solutions to unit vector size is necessary:

$$||\boldsymbol{b}_1||^2 = 1 \Leftrightarrow ||\boldsymbol{b}_1|| = 1.$$

Restricting the solution space results into a constrained optimization problem, given as

$$\begin{aligned} \max \ & \boldsymbol{b}_1^\top \boldsymbol{\Sigma} \boldsymbol{b}_1 \\ \text{s.t.} \ & ||\boldsymbol{b}_1||^2 = 1. \end{aligned}$$

Applying the method of Lagrange multipliers, the new objective function is obtained as

$$\mathcal{L}(\boldsymbol{b}_1, \lambda) = \boldsymbol{b}_1^\top \boldsymbol{\Sigma} \boldsymbol{b}_1 + \lambda_1(1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1).$$

The partial derivative of $\mathcal{L}$ with respect to $\boldsymbol{b}_1$ gives

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1} &= 2\boldsymbol{b}_1^\top \boldsymbol{\Sigma} - \lambda_1 \boldsymbol{b}_1^\top \\ 0 &= 2\boldsymbol{b}_1^\top \boldsymbol{\Sigma} - \lambda_1 \boldsymbol{b}_1^\top \\ \boldsymbol{\Sigma} \boldsymbol{b}_1 &= \lambda_1 \boldsymbol{b}_1. \end{aligned}$$

By the definition shown above, $\boldsymbol{b}_1$ has to be an eigenvector of $\boldsymbol{\Sigma}$ with the lagrangian multiplier as corresponding eigenvalue. Then, computing the derivative with respect to λ_1 results in

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \lambda_1} &= 1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1 \\ 0 &= 1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1 \\ \boldsymbol{b}_1^\top \boldsymbol{b}_1 &= 1. \end{aligned}$$

Finally, the variance objective from equation (2.23) can be rewritten as

$$\mathbb{V}_1 = \boldsymbol{b}_1^\top \boldsymbol{C} \boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1^\top \boldsymbol{b}_1 = \lambda_1. \tag{2.24}$$

The variance of the projected data equals the eigenvalue that is related to the basis vector $\boldsymbol{b}_1$, which spans the subspace the data is projected onto. In other words, the eigenvector that is associated with the biggest eigenvalue λ_1 is the first principal component.

Since the vector $\boldsymbol{b}_1^\top$ is considered a linear transformation, computing its inverse projects z_{1n} back into the original data space $\mathbb{R}^D$, such that

$$\tilde{\boldsymbol{x}}_n = \boldsymbol{b}_1 z_{1n} = \boldsymbol{b}_1 \boldsymbol{b}_1^\top \boldsymbol{x}_n.$$

So far, only the solution for the first principal component has been derived. Generalizing this schema onto a M-dimensional subspace with maximal variance involves some way of mathematical induction. The idea is to subtract the effect of the first $M - 1$ principal components $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{M-1}$ from the data and thereby trying to find the principal components that compress the remaining information. For a detailed investigation of this general case, reference is made to [DFO20, pp.291]

Chapter 3

State of the Art

3.1 Data Dissemination in Collaborative Intrusion Detection

3.2 Similarity Hashing in Malware Detection

3.3 Generative Algorithms and Intrusion Detection

Chapter 4

Generative Pattern Database

Three main challenges in the context of data dissemination in CIDS were identified. First, intrusion related data is usually of sensitive nature. Thus, the exchange mechanism must not compromise any policies and regulations related to data *privacy*. At the same time, the usability of the data has to be preserved. Second, the data that is subject of the exchange may exhibit large volumes. That constitutes a challenge, since the dissemination is desired to be executed with *minimal overhead* in a timely and scalable fashion. Lastly, the *interoperability* of the CIDS with existing local IDS is an important aspect that influences the practical adoption into security architectures. In summary, existing approaches for data dissemination mainly provide mechanisms for exchanging alert data or single attributes, e.g. IP addresses, as they focus on the correlation of intrusion detection incidents that originate from different sensors. The exchange of actual training data is neglected, possibly due to high data volumes. Thus, these systems lack of mechanisms for the extraction and global persistence of novel attack patterns, e.g. from zero day exploits, that can be used for the training of an intrusion detection sensor.

The approach that is presented in this chapter exchanges attack patterns by sharing generative machine learning models that have been trained on partitions of similar data points. Such a model-based dissemination enables the receiving side to sample a synthetic dataset that enhances existing local datasets. This provides two main advantages. First, no original data leaves a local network and therefore does not violate any privacy restrictions. Second, the data is compressed considerably by representing it in form of a generative model. In order to make that mechanism scalable, the monitored data is clustered using random projections. This way, similar data points are partitioned into globally common clusters, which is exploited as a data parallelism mechanism. Given that, bursty workloads can be served effectively in a cloud deployment. Furthermore, this mechanism enables a similarity-based correlation of distributed intrusion events. The integration of both a similarity based correlation of intrusion incidents and a mechanism for sharing attack knowledge makes it possible to extract novel patterns of distributed attacks and provide them globally within the CIDS, resulting in an improved attack detection.

Section 4.1 introduces the idea and key concepts of the approach. After that, Section 4.2 specifies the proposed architecture in detail. A comprehensive description of algorithms and processing steps are described in Section 4.3.

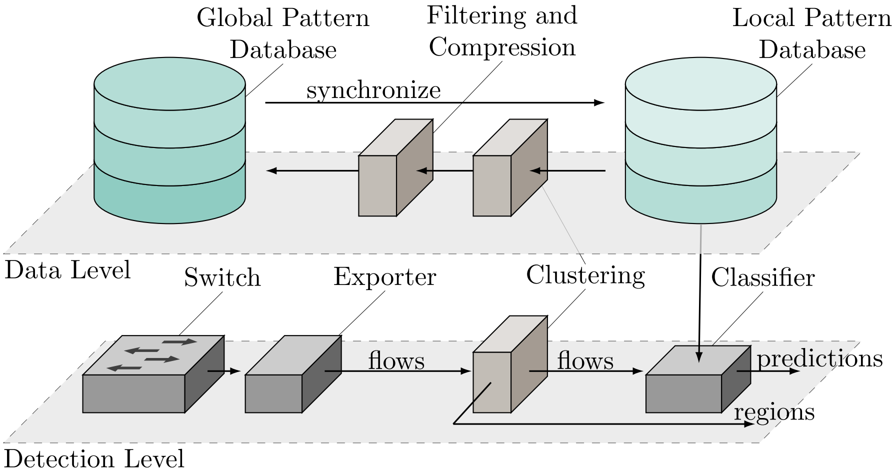


Figure 4.1: Example integration of the generative pattern database into a generic NIDS deployment, seperated by data and detection level

4.1 High Level Overview

Several members exist in the CIDS, each of which manages an isolated IDS. Every IDS operates according to a specific set of rules, that is essentially based on the content of a local database. The goal of the generative pattern database is to close the knowledge gaps of local databases and thus increase the detection rate of associated IDSs. By providing a *global view* on all local databases, individual IDSs can benefit from the collective knowledge of the CIDS.

Section 4.1.1 starts with a reference example to illustrate the idea that is described above and discusses the integration of the CIDS into existing infrastructures. Subsequently, Section 4.1.2 and Section 4.1.3 show the key concepts that enable data distribution and correlation under the given requirements. Finally, Section 4.1.4 combines the individual elements to present the strategy for the creation and utilization of the global view.

4.1.1 Example Integration

An exemplary integration of the CIDS into a generic NIDS is shown in Figure 4.1. The exemplary NIDS deployment consists of three components, that can be found on the detection layer. A flow exporter computes statistical flow features based on the network packets of a switch. A discriminative model serves as a classifier that operates on the specific feature set that the exporter extracts. After completing the training of a classifier instance on a given dataset, it is deployed within the detection pipeline. There, the classifier receives a stream of network flows and predicts them.

The CIDS mainly integrates at the data level, where the training data for the classifier is provided by the *local pattern database*. At this point, the local pattern database only contains the *local view* of the intrusion detection data from that particular member. In order to provide a global view, a synchronization process between its local pattern database and the *global pattern database* has to be initiated. Before the data is transferred to the global pattern database, it is subject to a set of clustering, filtering and compression operations. This way, data privacy is maintained and overhead is minimized by reducing the data volume.

On the receiving side, the global pattern database combines data from all members into a global view. Upon each update of the global pattern database, the new state of the global view is synchronized to each local pattern database and subsequently enhances the classifier on the detection level by extending the local intrusion detection dataset. Furthermore, an identical clustering operation as on the data level is applied on the detection level. Each incoming flow is assigned to a certain cluster, which is referred to as *region*. While the predictions from the classifier are suitable for detecting attacks that are known to the CIDS, regions are leveraged for the detection of novel data patterns and similarity-based correlations that uncover large-scale coordinated attacks, which are executed on the resources of multiple CIDS members simultanously.

4.1.2 Clustering

The clustering operation has to meet certain requirements in this approach.First, the results of the clustering operation should be consistent while it is executed among all members of the CIDS in a distributed fashion. Additionally, the algorithm should be scalable, since it is to be applied on whole databases on the data level and on streams of live data on the detection level. Given these requirements, gaussian random projection (GRP) (see Section 2.2.3) is selected. By using GRP, real data points are clustered according to their angular distance. Furthermore, the projection result is a binary string, which can be used for storing similar data points into a common bucket of a hash table by using the binary string as an index. In the context of the generative pattern database, the combination of GRPs and hash tables is exploited as the main controlling primitive for data persistence and retrieval. Instead of using randomly selected projection planes, a shared seed results in the application of a common projection function among all members of the collaboration. In other words, similar data points from different datasets are indexed to a common global bucket, i.e. region. Each region is subject to the transformations individually, which is utilized as a data parallelism mechanism. Thus, this approach is natively suited for cloud deployments where bursty workloads can be served effectively by balancing the load on an arbitrary number of instances. Lastly, this mechanism enables a similarity-based correlation of distributed intrusion events. As incoming data is monitored on the detection level, the clustering is applied, which results in a pattern that can be used for novelty checks or global occurrences within the CIDS.

4.1.3 Filtering and Compression

Two types of data are extracted within individual regions. First, metadata of local datasets is collected by counting label occurrences, which serve as indicator for determining if the respective region needs to be subject to the second extraction type. Second, models that are trained with generative algorithms on local attack data are the exchange medium for disseminating information within the CIDS. This provides two main advantages. For one, no original data leaves a local network and therefore does not violate any privacy restrictions. For another, the data is compressed considerably by representing it in form of a generative model.

4.1.4 Global View

As shown in Figure 4.2, each view is partitioned by a common projection function into an identical set of regions. Each local view contains original datapoints from the local dataset of the respective CIDS member. In this example, there exist three different classes globally, which occur differently in each local dataset. Examining a specific region, the combination of unique classes within all local views determines its complexity on a global level. For instance, the yellow region (global view) is simple, because within all local views, there occurs only a single class. A complex region in the global view, on the other hand, is formed by the occurrence of multiple classes within the same region in all local views. If a region in the global view contains more than one class, it potentially exhibits a non-linear decision boundary, hence it is called complex. This complexity information is distributed to all members of the CIDS by tagging the corresponding local regions with the complexity state of the region from the global view. After that, all local regions that are tagged as complex are modeled by a generative algorithm by using the corresponding local data points as training data. Subsequently, the resulting models are transferred to the global view. A synchronization process disseminates the region complexity estimations and the generative models to all local views. Within each local view, data points are sampled from generative models which are subsequently assembled into a synthetic dataset. Finally, the synthetic datasets are blended into the respective local datasets for enhancing the subsequent training of discriminative models.

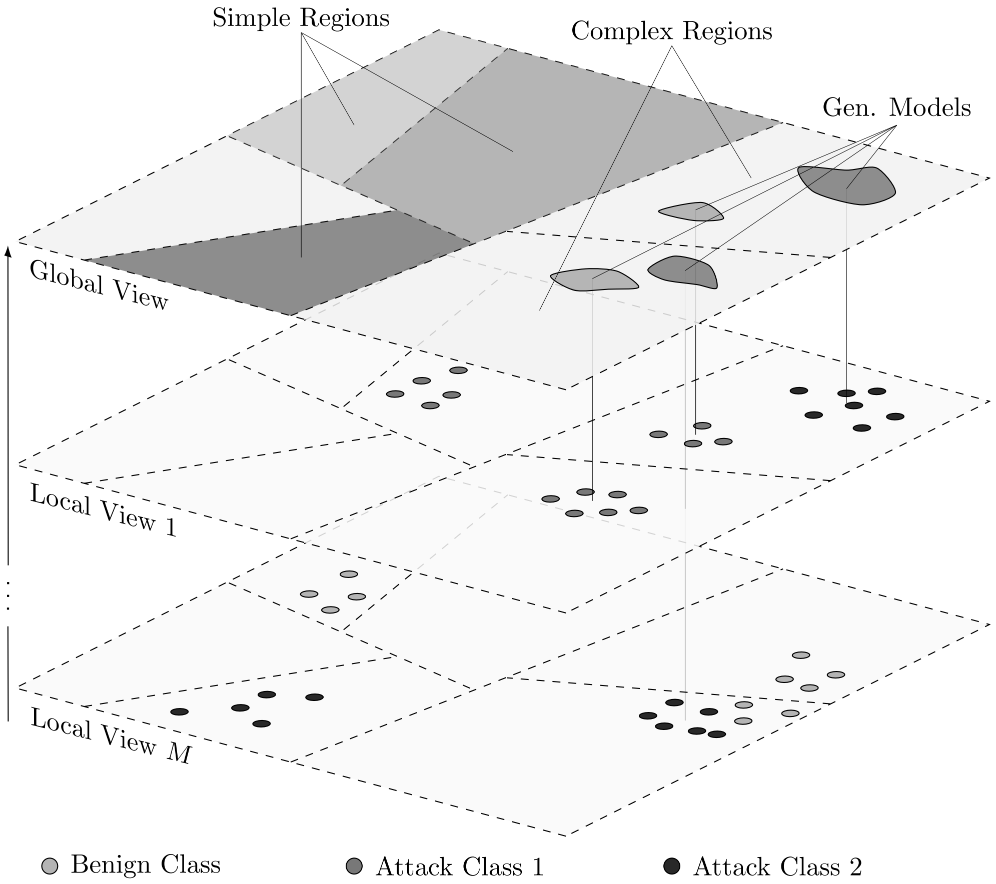


Figure 4.2: Building the global view by combining M local views

4.2 Architecture Specification

Logically, the proposed CIDS exhibits a hierarchical architecture (see Figure 4.3). For one, the global infrastructure G represents the collection of $M \in \mathbb{N}$ CIDS participants and their knowledge on an abstract level. For another, it provides specific services, that are globally available to each local infrastructure $L_m, m \in \{1, \dots, M\}$ that includes all CIDS components and services within the IT infrastructure boundaries of a corresponding CIDS member. Each local infrastructure L_m agrees to a specified feature extraction process that provides the monitoring data $\mathcal{X} \subset \mathbb{R}^D$ with a total number of features $D = |\boldsymbol{x}| \in \mathbb{N}$ for the attack detection. As the deployed NIDS is considered to be a supervised classification task, the ultimate goal is to find an approximation of an unknown target function $c : \mathcal{X} \to \mathcal{Y}$ that maps examples $\boldsymbol{x} \in \mathcal{X}$ to classes $y \in \mathcal{Y}$. In order to find an approximation $\hat{c} \sim c$, an individual training dataset $\mathcal{D}_m = \{(\boldsymbol{x}_n, y_n) : 1 \leq n \leq N_m\}$ of size $N_m = |\mathcal{D}_m| \in \mathbb{N}$ is provided in every L_m. True target values $c(\boldsymbol{x}) = y$ are given by the domain expert that assembled D_m. In addition, in the CIDS network, a common $\mathcal{Y}$ is agreed upon, so that there is a global consensus on class memberships.

CIDS communication across local boundaries occurs exclusively in a vertical direction. Thus, the exchange of information between individual L_m takes place indirectly via the global pattern database (PDB_G) and the global event channel (C_G). Each L_m includes a local pattern database (PDB_{L_m}), a local event channel (C_{L_m}) and an event-based data processing pipeline that consists of four services, namely *Local Indexing*, *Complexity Estimation*, *Generative Fitting* and *Classifier Fitting*.

Each instance of a pattern database (PDB) is realized as a key-value store and depending on the scope different tasks are considered. Each PDB_{L_m} is responsible for storing D_m and corresponding metadata. The PDB_G stores global metadata and the generative models that resemble the original data from each D_m. The notation for operations on a PDB corresponds to the notation for hash table operations defined in Section 2.2.2. Note, that if a locality-sensitive hash function is used to construct a key, in practice a hash table will internally apply a non-cryptographic hash function on the key to ensure an even distribution of the keys across the slots.

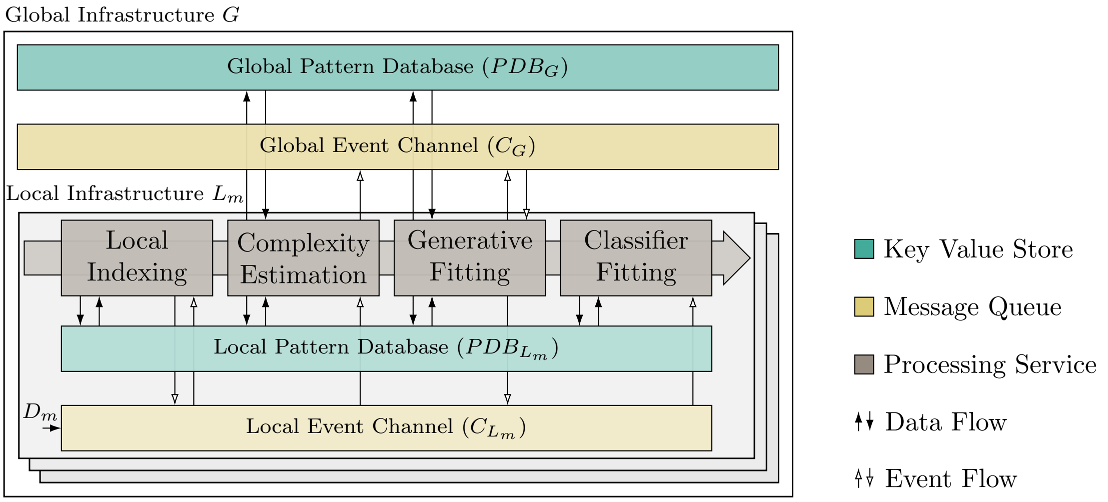

Global Infrastructure G
Global Pattern Database (PDB_G)
Global Event Channel (C_G)
Local Infrastructure L_m
Local Indexing
Complexity Estimation
Generative Fitting
Classifier Fitting
Local Pattern Database (PDB_{L_m})
D_m
Local Event Channel (C_{L_m})
Key Value Store
Message Queue
Processing Service
Data Flow
Event Flow

Figure 4.3: CIDS architecture illustrating the data and event flow between components within and across local infrastructure boundaries

Nonetheless, the clustering of neighbouring points into common slots is still ensured in this scenario. Each event channel C provides a topic-based publish-subscribe messaging mechanism that is mainly used to instantiate and distribute workloads among the service instances in the processing pipeline. Via the messaging system, service instances receive and emit events, on which upon the respective operations are triggered. Changes in a PDB result in responses that in turn are leveraged as the respective events. In this fashion, updates are propagated throughout the processing pipeline, ensuring a timely consistency among the pattern databases.

4.3 Service Specification

In Section 4.1, the roles of the clustering, filtering and compression operations within the architecture and the idea of the global view have been roughly explained. These concepts are implemented within the processing pipeline that has been referred to in Section 4.2 as four consecutive and event-based services, which are described in detail by focusing on a single service in each subsection.

4.3.1 Local Indexing

The local indexing service is responsible for the preprocessing and local storage of D_m. As already described in Section 4.1.2, the data is organized in regions. This means that individual data points are first assigned to a region using a GRP. By doing that, a hash of the data point is computed. Based on the generated hash value, a key is constructed that is used to persist that data point in the respective PDB_{L_m}. According to the properties of a locality-sensitive hash function, similar data points are assigned to a common region and thus form a closed processing unit for subsequent operation steps. If a region is initialized or an update is made to an existing region, e.g., due to the occurrence of new data, events are emitted to inform the subsequent service.

First, an intrusion detection dataset D_m is sent to one or more service processors via the C_{L_m}. Second, the incoming stream of pairs of data points and labels $(\boldsymbol{x}_n, y_n) \in D_m$ is ingested and buffered until a batch $X = (\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_B, y_B)$ of size $B \in \mathbb{N}$ has been accumulated. Then, X is preprocessed and inserted into the PDB_{L_m} as described in Algorithm 5. In words, a flow-feature vector $\boldsymbol{x}$ is scaled to the range $[-1, 1]$ by applying a feature-wise min-max normalization given by

$$\boldsymbol{x}' = (b - a)\,\frac{\boldsymbol{x} - \min \boldsymbol{x}}{\max \boldsymbol{x} - \min \boldsymbol{x}} + a \tag{4.1}$$

where $a = -1$ and $b = 1$. After that, the scaled data point $\boldsymbol{x}'$ is subject to both a locality-sensitive hashing function h and a non-cryptographic hashing function g. In this particular architecture, h is a GRP with a global seed for the initialization of the projection plane $\boldsymbol{M}$ (see Section 2.2.3), such that regions $r = h(\boldsymbol{x}')$ across local infrastructures are comparable and consistent.[1] Since the data is organized in regions, a nested scheme is applied for the insertion of pairs of datapoints and labels as depicted in Figure 4.4.

[1] Here, the words hash value and region are used synonymously

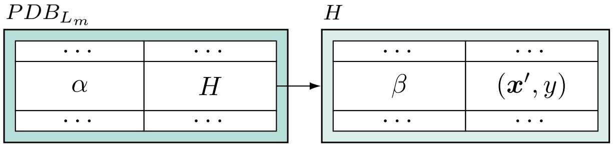


Figure 4.4: Nested indexing in a local pattern database.

In fact, the pairs within a region are further partitioned into disjoint subsets according to the true class label y. This means that for each subset of the data of a particular label within a region, a separate hashtable is initialized and inserted into the PDB_{L_m} as a second level. Thus, for the persistence of a pair $(\boldsymbol{x}', y)$, two keys α, β are constructed (see Lines 6-7 in Algorithm 5). The key α is a concatenation of the prefix constant p_x, the bit-string r and the label y. This way, the data is partitioned primarily by its region and secondarily by its label as described above. The key β is the result of the non-cryptographic hashing function $g(\boldsymbol{x}')$, which serves as a mechanism for deduplicating identical $\boldsymbol{x}'$. Additionally, a third key γ is constructed by concatenating the prefix constant p_y and the bit-string r. As it is important to retrieve all existing labels within a region efficiently in a processing step of the subsequent service, γ is used for storing the set of labels within a region as auxiliary metadata. Next, if not already present, a hash table is initialized and inserted into PDB_{L_m} using α. Likewise, if PDB_{L_m} at slot γ is empty, a new set[2] is initialited and inserted. After that, it is checked if the slot β of the hash table placed at α in PDB_{L_m}, is empty. In the positive case, the pair $(\boldsymbol{x}', y)$ is inserted into that slot and the region r is inserted into the set $\mathcal{R}$. Otherwise, no data is inserted into the local pattern database and no region is added to $\mathcal{R}$. After processing a batch, the set of updated regions $\mathcal{R}$ is emitted as events into C_{L_m}.

```
Input: Batch $X$
Output: Regions $\mathcal{R}$
 1: $\mathcal{R} \leftarrow$ new Set
 2: $\min \boldsymbol{x}, \max \boldsymbol{x} \leftarrow$ extract feature ranges from $X$
 3: for each $(\boldsymbol{x}, y)$ in a batch do
 4:     $\boldsymbol{x}' \leftarrow$ normalize$(\boldsymbol{x})$                    ▶ see Equation 4.1
 5:     $r \leftarrow h(\boldsymbol{x}')$
 6:     $\alpha \leftarrow$ concatenate$(p_x, r, y)$
 7:     $\beta \leftarrow g(\boldsymbol{x}')$
 8:     $\gamma \leftarrow$ concatenate$(p_y, r)$
 9:     if $PDB_{L_m}[\alpha]$ is None then
10:         $PDB_{L_m}[\alpha] \leftarrow$ new hash table $H$
11:     if $PDB_{L_m}[\gamma]$ is None then
12:         $PDB_{L_m}[\gamma] \leftarrow$ new set $\mathcal{S}_r$
13:     if $PDB_{L_m}[\alpha][\beta]$ is None then
14:         $PDB_{L_m}[\alpha][\beta] \leftarrow (\boldsymbol{x}', y)$
15:         insert $y$ into the set at $PDB_{L_m}[\gamma]$
16:         insert $r$ into $\mathcal{R}$
17: return $\mathcal{R}$
```

Algorithm 5: Preprocessing and inserting $B \subset D_m$ into PDB_{L_m}

[2] A set is a data structure describing an unordered collection with no duplicate elements.

4.3.2 Complexity Estimation

As described in Section 4.1.4, a region is said to be complex if it contains more than one unique label. Otherwise, a region is simple. Since the data within a region already represents a cluster created by a locality-sensitive hashing function h, the existence of multiple classes within a single region indicates a non-linear decision boundary. On that basis we differentiate how a region is processed in the subsequent services of the pipeline. Furthermore, the complexity state of a region may vary, depending on the scope it is observed. Note that since the projection matrix $\boldsymbol{M}$ that is used to construct h is initialized with the same values in every L_m, all hashes that were computed by h are globally comparable. This means that similar data points from different datasets, e.g., $\boldsymbol{x}_i \in D_1$ and $\boldsymbol{x}_j \in D_2$, $\boldsymbol{x}_i \sim \boldsymbol{x}_j$ may be hashed to the same region $h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)$. However, it is also possible that that the corresponding labels $y_i \in D_1$ and $y_j \in D_2$ are not equal. Thus, a certain region may contain only a single unique label in both local scopes but multiple unique labels in the global scope and therefore lead to a different global view on that region's complexity state. Given that, the complexity estimation service acts as a bridge between the local and global components and answers the question, which regions are considered to be complex in a global scope.

First, a set of incoming regions $\mathcal{R}_{\text{in}}$ is received on the C_{L_m}. Then, for each region $r \in \mathcal{R}_{\text{in}}$ the following operations are executed (see Algorithm 6). The set of unique labels $\mathcal{S}_r$ for a particular region within a local scope has been stored in Algorithm 5 as auxiliary metadata, which is now retrieved from the PDB_{L_m} by constructing the correspoding key γ. In the next step, $\mathcal{S}_r$ has to be stored in the PDB_G. Therefore, another key δ is constructed by concatenating the prefix constant p_y, the region r and the current local infrastructure identifier m, which prevents the collision of information from different infrastructures. After $\mathcal{S}_r$ is stored on a global level at $PDB_G[\delta]$, the label set information for that region from all members in the CIDS is aggregated. That aggregated view is essentially the global complexity state for that region. By iterating over all member identifiers in the CIDS, multiple keys δ are constructed. Each key retrieves the specific label set $\mathcal{S}_r$ of a member and inserts its content into the temporary set $\mathcal{S}_t$, collecting the local label sets.

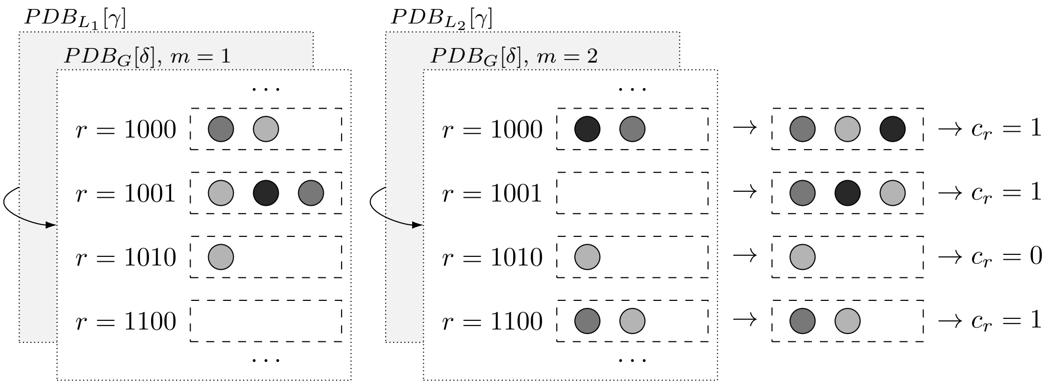


Figure 4.5: Illustration of the complexity estimation algorithm with two local infrastructures. Each local label set at $PDB_{L_m}[\kappa]$ is synchronized into the global pattern database at $PDB_G[\delta]$, where δ is constructed with m. Per region, the union operation is applied on the global label sets, whereupon the complexity is derived from.

After collecting all label sets in $\mathcal{S}_t$, the complexity state is obtained by simply evaluating the cardinality $|\mathcal{S}_t|$. If there is more than one class in a region on a global scope, that is $|S| > 1$, then assign a true value to the global complexity variable c_r. Otherwise, assign a false value. In order to store c_r, the key κ is created by concatenating the prefix constant p_c and the region r. Note, that $PDB_G[\kappa]$ is only updated, if storing c_r changes the state that is already persisted. This is because, if an update is executed, this information has to be propagated to the next service. Thus, in that case, the region r is inserted into $\mathcal{R}_{\text{out}}$, which is subsequently sent into the global event channel C_G in order to inform services in all local infrastructures about the update.

```
Input: Regions $\mathcal{R}_{\text{in}}$
Output: Regions $\mathcal{R}_{\text{out}}$
$\mathcal{R}_{\text{out}} \leftarrow$ new Set
$m \leftarrow$ getID()                         ▶ current local infrastructure identifier
for each $r$ in $\mathcal{R}$ do
    $\gamma \leftarrow \text{concatenate}(p_y, r)$
    $\mathcal{S}_r \leftarrow PDB_{L_m}[\gamma]$
    $\delta \leftarrow \text{concatenate}(p_y, r, m)$
    $PDB_G[\delta] \leftarrow \mathcal{S}_r$
    $\mathcal{S}_t \leftarrow \emptyset$
    for each $m$ in $\{1, \ldots, M\}$ do
        $\delta \leftarrow \text{concatenate}(p_y, r, m)$
        $\mathcal{S}_t \leftarrow \mathcal{S}_t \cup PDB_G[\delta]$
    if $|\mathcal{S}_t| > 1$ then
        $c_r \leftarrow 1$
    else
        $c_r \leftarrow 0$
    $\kappa \leftarrow \text{concatenate}(p_c, r)$
    if $PDB_G[\kappa] \neq c_r$ then
        $PDB_G[\kappa] \leftarrow c_r$
        add $r$ to $\mathcal{R}_{\text{out}}$
return $R_{\text{out}}$
```

Algorithm 6: Construction of a global complexity state

4.3.3 Generative Fitting

The generative fitting service is the most demanding procedure in the context of processing resources. The service represents the filtering and compression operations presented in Section 4.1. There are two scenarios based on the region's complexity. If the region is not complex, no further actions are taken except it formally was complex. Then, existing models have to be deleted, since they are not longer used. And if the region is complex, generative models are provided, which represent the medium for exchanging information on attacks. More specifically, multiple GMM are fitted on each label-subset of a region's data, which was stored in the indexing step in Section 4.3.1. According to a model selection process that evaluates the efficacy of each GMM, the best model is stored in the global pattern database, accessible to every member in the CIDS to sample synthetic data from.

This way, every member has access to the global knowledge from all local infrastructures in order to enhance the local dataset that is used for fitting a classifier. Thus, this service is the key for providing *privacy* and *minimal overhead* while exchanging information. Considering the length and complexity of the complete algorithm, the main procedure is outlined first in Algorithm 7 and then the details on important sub-routines are elaboreted subsequently.

Regions that have been updated in the preceding service are received as events $r \in \mathcal{R}_{\text{in}}$. Since the data is further organized per label within a region, the label set for a region $\mathcal{S}_r$ is retrieved. Additionally, the complexity state of the region c_r is checked. Then, if the region is not complex, no model fitting is executed. Instead, potentially existing models are deleted from storage. This is the case, if the complexity status of the region has been changed from complex to simple. But if the region is currently complex, the generative model fitting procedure is triggered where new models are created and already existing models are updated. For that, from every hash table within a region, the original flow samples are extracted and collected in $\mathcal{X}_\alpha$, which is then stored in T with the corresponding label as the key. The goal is to fit a model on each $\mathcal{X}_\alpha$ within a region, such that the collection of models for a region can be used in combination for sampling the synthetic data. Note that this is a specific process for the employed generativ algorithm. Since GMMs cannot be conditioned on multiple labels.

```
Input: Regions $\mathcal{R}_{\text{in}}$
Output: Regions $\mathcal{R}_{\text{out}}$
$\mathcal{R}_{\text{out}} \leftarrow \emptyset$
for each $r$ in $\mathcal{R}_{\text{in}}$ do
    $\kappa \leftarrow$ concatenate($p_c, r$)
    $\delta \leftarrow$ concatenate($p_y, r, m$)
    $c_r \leftarrow PDB_G[\kappa]$
    $\mathcal{S}_r \leftarrow PDB_G[\delta]$
    if $c_r = 0$ then
        for each $y$ in $\mathcal{S}_r$ do
            $\omega \leftarrow$ concatenate($p_d, r, y, m$)
            delete model in $PDB_G[\omega]$
    else
        $T \leftarrow$ new hash table
        for each $y$ in $\mathcal{S}_r$ do
            $\alpha \leftarrow$ concatenate($p_x, r, y$)
            $\mathcal{X}_\alpha \leftarrow$ extract values (flows) from $PDB_{L_m}[\alpha]$
            $T[y] \leftarrow \mathcal{X}_\alpha$
        $T \leftarrow$ upsampling($T$)
        for each $(y, F_y)$ in $T_F$ do
            GMM $\leftarrow$ modelSelection($y, F_y$)
            $\omega \leftarrow$ concatenate($p_d, r, y, m$)
            $PDB_G[\omega] \leftarrow$ GMM
        add $r$ to $\mathcal{R}_{\text{out}}$
return $\mathcal{R}_{\text{out}}$
```

Algorithm 7: Generative Fitting (Main Procedure)

Note that this is a specific process for the employed generative algorithm. Since GMMs cannot be conditioned on multiple labels, multiple single label models have to be fitted. Subsequently, upsampling operations prepare the collected region data in T for the model fitting (see Algorithm 8). After that, the model selection process is started sequentially for each available label in the region (see Algorithm 9). Finally, the best fitted model is stored in the PDB_G.

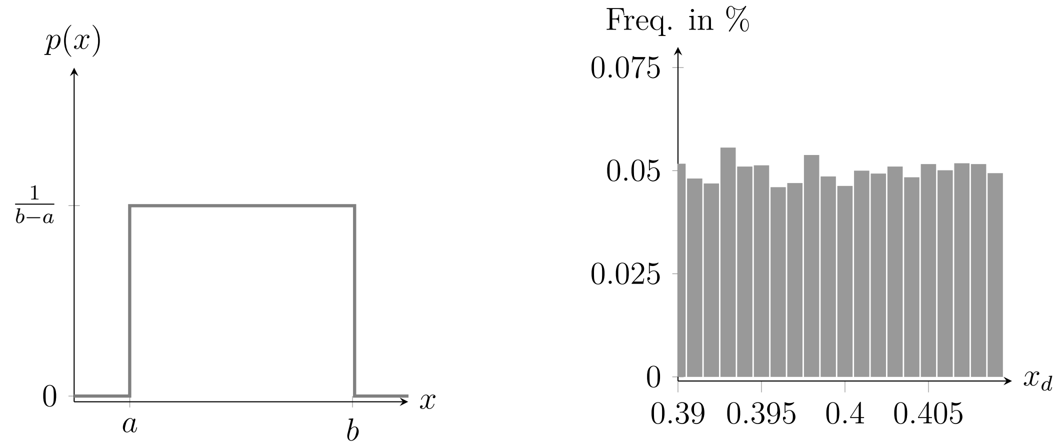
So far, the main procedure has been outlined. Next, the details on the upsampling are elaborated (see Algorithm 8). As each $\mathcal{X}_\alpha$ is used for the fitting of a GMM and since some of these flow data subsets of a region potentially exhibit a low number of samples, an upsampling process may need to applied. The upsampling process ensures that the fitting algorithm for the GMM is able to work. For that, the number of samples has to be at least equal to the number of components of the GMM which in turn is at most equal to the number of dimensions of the training dataset $\mathcal{X}_\alpha$. And since different parameters for the number of components are tried during the model selection, the upper limit for the parameter is used as a comparison value. In this case the upper limit for the number of components is the number of dimensions of the training data. If this is not the case, the upsampling fills the lacking samples artificially.

For the initialization of the responsibilities of the mixture components within the fitting procedure of the GMM the k-means algorithm is used to find distinct clusters within $\mathcal{X}_\alpha$. The resulting cluster assignments for each $\boldsymbol{x} \in \mathcal{X}_\alpha$ are then used as initial values for the responsibilities. A supersampling process results in duplicate points that could lead to a smaller number if distinct clusters found by the k-means algorithm than the number of components specified in the model. Therefore, the upsampling is realized by generating near neighbours per sample.

First, the number of neighbours to generate per sample N_Δ is determined by dividing the difference of the dimensionality of the flow data $\dim(\mathcal{X}_\alpha)$ and the number of data points $|\mathcal{X}_\alpha|$ by $|\mathcal{X}_\alpha|$ using an integer division. This ensures that each original data point $\boldsymbol{x}$ contributes an equal share to the total amount of newly sampled points $\hat{\boldsymbol{x}}$. In case, the result of the integer division is zero, one is added to the result. Then for each data point $\boldsymbol{x} = [x_1, \ldots, x_D]^\top$, nearest neighbours are generated by sampling from a uniform distribution $\mathcal{U}(a, b)$.

```
Input: Hash table $T$ with labels $y$ as keys and sets of flows $\mathcal{X}$ as values
Output: Hash table $T$ with upsampled sets of flows
for each $(y, \mathcal{X}_\alpha)$ in $T$ do
    if $y \neq 0$ and $|\mathcal{X}_\alpha| < \dim(\mathcal{X}_\alpha)$ then
        $\hat{\mathcal{X}} \leftarrow \emptyset$
        $N_\Delta \leftarrow \lfloor \dim(\mathcal{X}_\alpha) - |\mathcal{X}_\alpha| \,/\, |\mathcal{X}_\alpha| \rfloor + 1$
        for each $\boldsymbol{x}$ in $\mathcal{X}_\alpha$ do
            for each $n$ in $\{1 \leq n \leq N_\Delta\}$ do
                $\hat{\boldsymbol{x}} \leftarrow [s(x_1), \ldots, s(x_D)]^\top$ with $s(x_d) \sim \mathcal{U}(a, b)$
                add $\hat{\boldsymbol{x}}$ to $\hat{\mathcal{X}}$
        $\mathcal{X}_\alpha \leftarrow \mathcal{X}_\alpha \cup \hat{\mathcal{X}}$
        $T[y] \leftarrow T[y] \cup \mathcal{X}_\alpha$
return $T$
```

Algorithm 8: Binary Splitting and Upsampling of Region Data



(a) The pdf of a uniform distribution is $p(x) = \frac{1}{b-a}$ within the interval $[a, b)$, and zero elsewhere.

(b) Histogram of 1 000 samples drawn uniformly over the interval $[x_d - \epsilon, x_d + \epsilon)$ where $x_d = 0.4$ and $\epsilon = 1 \cdot 10^{-2}$.

Figure 4.6: Nearest neighbours are generated by sampling each feature value from a the uniform distribution.

That way, the nearest neighbour of $\boldsymbol{x}$ is the concatenation of the sampled feature values as $\hat{\boldsymbol{x}} = [s(x_1), \ldots, s(x_D)]^\top$, where $s(x_d) \sim \mathcal{U}(a, b)$ is defined as the value sampled from the uniform distribution within the interval $[x_d - \epsilon, x_d + \epsilon)$. The value ϵ controls the size of the interval of the uniform distribution. The larger the value for ϵ, the further the newly generated values deviate from the orginal feature values. In Figure 4.6b the value for a single feature x_d is drawn uniformly at random. The original value of the feature was $x_d = 4$, which was extended by $\epsilon = 1 \cdot 10^{-2}$. By choosing a relatively small value for ϵ, it is ensured that the generated nearest neighbours do not alter the original data distribution significantly, while enabling a more ideal fitting of the GMM for that set of data points. Finally, the generated neighbours and the original data points are combined. After processing each pair $(y, \mathcal{X}_\alpha)$, the hash table T is returned to the main procedure (Algorithm 7) for the subseqeuent model selection.

In the next phase of the algorithm, one GMM is selected per unique label within a region. During the selection process multiple models are fitted with different model parameters using a flow data subset $\mathcal{X}_\alpha$. Since the resource demands of the EM Algorithm for fitting a GMM are relatively high, the dimensionality of the training data is reduced by applying a dimensionality reduction via PCA. Later in the pipeline, when sampling data from the GMMs, the inverse operation of the corresponding PCA is applied on the synthetic data in order to restore the original feature space. Therefore, the parameters of the PCA have to be stored in the PDB_G along with the parameters of the GMM.

Note, that during the application of PCA in the first place, information within the data is lost by only keeping the components that describe most of the variance of the data. Of course, the result of the inverse transformation would then only be an approximation of the original data. However, the input for the inverse transformation is going to be the synthetic data. Thus, we accept the loss and focus on the fact that the synthetic data is transformed back into the original feature space, such that the synthetic data resembles more the original data as it was captured, increasing its compatibility.

The model selection is specified in Algorithm 9. First PCA is applied on $\mathcal{X}_\alpha$ resulting in a set of flows $\mathcal{X}'_\alpha$ with a reduced feature set. The number of components of the PCA has to be set heuristically or a threshold has to be set, which defines the minimum amount of variance that needs to be explained by a specific number of components. After the compression, the parameter search for the GMM is started. In general, there is no exact method to determine the optimal parameters for a GMM, such that different parameter settings have to be evaluated. For that, the search space is defined by two dimensions, namely the number of components and the type of covariance matrix that is used for each component. The set of numbers that is used for specifying the number of components is defined as $\mathcal{C} = \{4c \,|\, c \in \mathbb{N}, 1 \leq c \leq \dim(\mathcal{X}_1)\}$. The covariance matrix can either be a standard covariance matrix with a full set of entries or can be an approximation with entries only on the diagonal of the matrix, i.e., we define $\mathbf{\Sigma} = \{\text{“full”}, \text{“diagonal”}\}$. The reason behind the choice of both types of covariance matrix is also motivated by heuristical optimization. The full covariance matrix can result in overfitting, especially on small datasets, whereas the diagonal matrix acts as a type of regularization to the model. However, depending on the data the diagonal type sometimes also leads to an underfitting of the model.

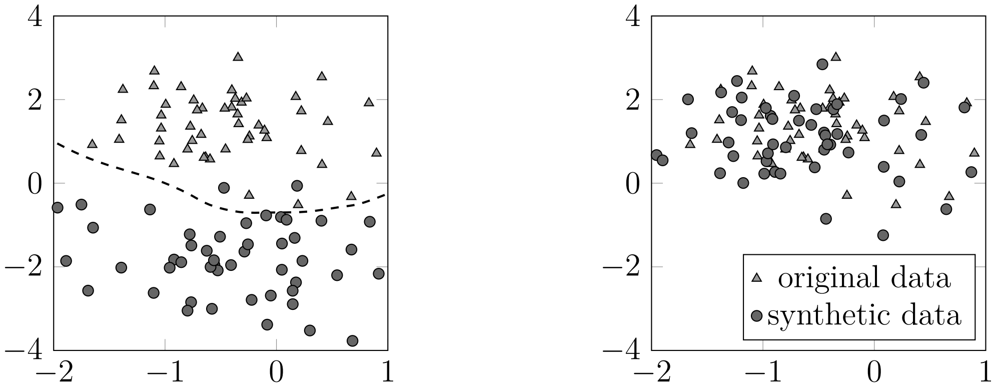
After fitting a GMM on $\mathcal{X}'_1$ the first metric for the selection is calculated. Precisely, the BIC is calculated as in Equation 2.20 (see Section 2.3.3). Note, that for the case of a diagonal covariance matrix, the number of entries of the covariance matrix to estimate changes from $\frac{D(D+1)}{2}$ to D. Thus, given a GMM with K mixture components, each exhibiting a diagonal covariance matrix, and a datasset $\mathcal{X}_1$ consisting of N data points and D features, the BIC is defined as

$$\mathrm{BIC}(\mathrm{GMM}|\mathcal{X}_1) = (KD + D + K - 1)\ln(N) - 2\ln(\hat{L}). \tag{4.2}$$

After that, a second evaluation metric is calculated (see Algorithm 10). The process that is used to gather that metric is called adversary evaluation and is based on the principles of Generative Adversarial Networks (GANs).

```
Input: Dataset $\mathcal{X}_1$
Output: Tuple ($GMS$, GMM, PCA)
$\mathcal{X}'_1 \leftarrow \mathrm{PCA}(\mathcal{X}_1)$
$\mathcal{C} \leftarrow \{4c \,|\, c \in \mathbb{N}, 1 \leq c \leq \dim(\mathcal{X}_1)\}$
$\mathbf{\Sigma} \leftarrow \{\text{“full”}, \text{“diagonal”}\}$
$L \leftarrow$ new list
for each $c$ in $\mathcal{C}$ do
    for each $\Sigma$ in $\mathbf{\Sigma}$ do
        $\mathrm{GMM} \leftarrow \mathrm{fitGaussianMixture}(\mathcal{X}'_1, c, \Sigma)$
        $\mathrm{BIC} \leftarrow \mathrm{BIC}(\mathrm{GMM}, \mathcal{X}'_1)$
        $ACC \leftarrow \mathrm{AdversaryEvaluation}(\mathcal{X}_1, \mathrm{GMM}, \mathrm{PCA})$
        $GMS \leftarrow GMS(\mathrm{BIC}, ACC)$
        add ($GMS$, GMM, PCA) to $L$
sort $L$ according to the values of $GMS$
return ($GMS$, GMM, PCA) with the lowest value for $GMS$
```

Algorithm 9: Model Selection



(a) A poor fitted generative model: The original data (triangles) and the synthetic data (circles) can be easily separated by a classifier.

(b) A well fitted generative model: The synthetic data resemble the original data well, such that a separation is not easy.

Figure 4.7: Adversary evaluation: seperate original and fake data.

Fitting a Generative Adversarial Network (GAN) involves a discriminator function, i.e., classifier that tries to distinguish between the original and synthetic data. In other words, the generative algorithm synthesizes fake data, which resembles the original data. Based on that, a dataset is assembled by combining fake data labeled as the positive class (1) and original data labeled as the negative class (0) or vice versa. This dataset is shuffled and split into a training and test proportion.

Then, a discriminative model is trained on the training proportion, trying to find distinguishing features within both classes, allowing a distinction to be made. Within the evaluation step, the discriminator is tested on its ability to separate fake and original data. The lower the score of the discriminator, the better the generative model. Figure 4.7 illustrates two scenarios. In the first scenario (a), the generative model has been fitted poorly, because the synthetic data can be easily separated from the original data as indicated by the decision boundary (dashed line). In the second scenario (a), the generative model performs well by creating synthetic data that could not be seperated from the original data.

```
**Input:** $\mathcal{X}_1, \mathrm{GMM}, \mathrm{PCA}$
**Output:** $ACC$ from decision tree model $DT$
$\tilde{\mathcal{X}}_1' \leftarrow$ sample $|\mathcal{X}_1|$ data points from GMM
$\tilde{\mathcal{X}}_1 \leftarrow \mathrm{PCA}^{-1}(\tilde{\mathcal{X}}_1')$
$\mathcal{Y} \leftarrow \{1\}$
$\tilde{\mathcal{Y}} \leftarrow \{0\}$
$\mathcal{D} \leftarrow \mathrm{concatenate}(\mathcal{X}_1 \times \mathcal{Y}, \tilde{\mathcal{X}}_1 \times \tilde{\mathcal{Y}})$
$\mathcal{D}_{\mathrm{train}}, \mathcal{D}_{\mathrm{test}} \leftarrow \mathrm{shuffleSplit}(\mathcal{D})$
$\mathrm{DT} \leftarrow \mathrm{fitDecisionTree}(\mathcal{D}_{\mathrm{train}})$
$\hat{\mathcal{Y}} \leftarrow$ predict $\mathcal{X}_{\mathrm{test}} \subset \mathcal{D}_{\mathrm{test}}$ with DT
$ACC \leftarrow ACC(\mathcal{Y}_{\mathrm{test}}, \hat{\mathcal{Y}})$
**return** $ACC$
```

Algorithm 10: Adversary Evaluation

In order to quantify the performance of the discriminator within the adversary evaluation, a classification metric is calculated. Since the original and synthetic data is equal in size and the posed problem is binary the accuracy score (ACC) is chosen, which is defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

For each iteration in the model selection process, both the BIC and the ACC are stored. After all parameter combinations have been exhausted and all resulting models have been evaluated a combined metric that we call generative model selection (GMS) is calculated from the BIC and ACC for each model. For that, the value range of both metrics must be aligned, such that the BIC is normalized into the range $[0, 1]$ by applying the min-max-normalization from Equation 4.1. After that, the mean is calculated for each metric pair as defined in Equation 4.4. Finally, the model with the lowest combined GMS is selected.

$$GMS = \frac{1 - ACC + \text{BIC}}{2}. \tag{4.4}$$

The model parameters of both the GMM and the PCA are stored in the PDB_G. In numbers, these can range, depending on which type of covariance matrix is chosen for the GMM. PCA on the other hand, requires a $D \times M$ projection matrix, with D being the dimensions of the original data space and M being the number of dimensions in the projected subspace. Note that $|\mathcal{S}_y|$ defines the number of unique labels within a region, which effectively determines the number of model pairs (GMM, PCA) that are to be stored per region. Therefore, per region, the number of parameters to store is at least

$$|\mathcal{S}_y| \left(2KD + K - 1 + MD\right)$$

and at most

$$|\mathcal{S}_y| \Big((KD + K \, \frac{D(D+1)}{2} + K - 1) + MD\Big).$$

4.3.4 Classifier Fitting

4.4 Summary

Chapter 5

Experimental Evaluation

5.1 Flow Feature Analysis

First the employed datasets and the included attacks are presented shortly at the beginning; After that, the feature extraction process is described. Thereby, it is explained, which kind of flow features are extracted and which flow exporter is used; The Preprocessing steps include the labeling of the extracted feature vectors and the splitting into the respective training and testing proportions; the distribution of the different classes within each dataset is described extensively for transparency; a combined analysis of the feature importances for all datasets shows that no further feature selection would be beneficial, which finalizes the data description.

5.1.1 Datasets

In the evaluation, multiple IT-infrastructures that are members of the cids are simulated by incorporating different benchmark datasets; The three following datasets are selected, which are from now on referred to by the corresponding roman numeral:

I CSE-CIC-IDS2018 [Com18],

II CIC-IDS2017 [SHG18],

III CIC-DoS2017 [Jaz+17].

When selecting the datasets, it was ensured that the attack classes contained in each case were similar or the same in order to be able to carry out an evaluation in the context of a collaboration.

Both the CIC-IDS-2018 and CIC-IDS-2017 are very alike from the included scenarios and attacks; they are designed as a diverse and comprehensive benchmark dataset; they are different from the utilized testbed; IDS 2017 attack network with four workstations, victim network three servers and ten workstations IDS 2018 used an attack infrastructure consisting of 50 machines and an victim infrastructure consisting of 420 machines and 30 servers; DoS2017 is an intrusion detection dataset that contains, beneath benign traffic only traces of application layer DoS attacks; instead of carrying out the DoS on the network layer using a SYN Flood, for example, the malicious client sends traffic either to a listening UDP socket or through a completed TCP connection; 8 different attack tools were used for executing the attacks, attacking 10 web servers; generally, the attacks can be categorized into high and low volume attacks; high volume attacks are characterized by a high volume of application-layer requests, which are transmitted to the victim; in contrast, low volume attacks are characterized by a small volume of attack traffic that has an impact on a service

sending a malicious pdf via an email to a victim; upon the usage of the file, an application (pdf reader) vulnerability is exploited; after the successful exploitation, a backdoor to the systems of the victim is executed, such that that systems is used for internal network scans (service enumeration using nmap, ip sweep and full range scan of ports using portscan)

tries multiple usernames and passwords, test a wide range of combinations (dictionary attack with 90 million entries), until they find the correct login information; it is not described if the login names are known beforehand ftp-bruteforece using patator - FTP login ssh-patator - SSH Login

Exploit of the OpenSSL cryptography library; sending malformed heartbeat request allows to read encrypted communication; steal secret keys for certificates; user names and passwords, instant messages

5.1.2 Feature Extraction and Preprocessing

In the context of network intrusion detection, network flow data is employed for real time network analysis. Within the context of real time flow feature analysis it is often neglected that flows are exported upon termination of the corresponding communication, which is either triggered by an activity or inactivtiy timeout mechanism. The specific setting of the timeout values is often a tradeoff between low data volume with a high timeout setting on the one side and a fine-grained analysis with low timeout values on the other side. However, especially in the area of network intrusion detection, a low detection latency heavily depends on timely available data. Thus, for a realistic evaluation, only early statistical flow features on the first 10 packets of a communication are considered.

A flow exporter that is implemented using the Python Framework NFStream is used for tracking and exporting bidirectional network flows. A total of 45 flow features are extracted, including the basic 5-tuple and various statistical values based on the number of packets, packet sizes, and inter-arrival times. An active and inactive timeout of 15s is specified for the flow export. All datasets presented in Section **??** are available in the pcap capture file format. Files from the datasets CIC-IDS-2017 and CIC-DoS-2017 were exported directly. In the case of the CIC-IDS-2018 dataset further preparations were required. Some files contained broken headers, which were fixed with the pcapfix repair tool. In addition, the individual days, by which the dataset is structured, were fragmented further into many individual files, which were merged into single files using the pcapmerge tool. Subsequently, these files were exported. The number of the resulting flow samples are shown in Table 5.1.

The exported data flow samples are labeled based on basic attack knowledge (timestamps, IP addresses) from the respective website from which the datasets can be accessed. Improvements of the labeling process regarding corrected timestamps and further sanity checks, which are described in [**engelen2021**], have been taken into account. Table **??** shows the number of samples per class of each respective dataset. For the chosen evaluation strategy, the dataset is further preprocessed by grouping and renaming the samples that represent denial of service attacks. More precisely, already existing classes, that are labeled after the name of the specific tool that was used to create that attack, are relabeled by a strategy that is depicted in Table **??**. Furthermore, specific attacks that are either only represented in a single dataset (e.g. FTP brute force) or are too specific to be compared to (e.g. various infiltration attacks) are removed. The resulting classes and corresponding sample counts are shown in Figure **??**.

permutation importance [Lou14, p. 125] mean decrease accuracy (MDA) or, equivalently, the mean increase rror, of the forest when the values of a column of the dataset, i.e. feature, are randomly permutet in the out-of-bag-samples "Permutation Importance or Mean Decrease in Accuracy (MDA) is assessed for each feature by removing the association between that feature and the target. This is achieved by randomly permuting the values of the feature and measuring the resulting increase in error. The influence

Dataset	Day / Filename	No. Flows
CIC-IDS-2018	Friday-02-03-2018	9 502 494
	Friday-16-02-2018	8 858 574
	Friday-23-02-2018	9 419 691
	Thursday-01-03-2018	10 392 041
	Thursday-15-02-2018	8 431 812
	Thursday-22-02-2018	9 629 536
	Tuesday-20-02-2018	9 261 384
	Wednesday-14-02-2018	8 948 042
	Wednesday-21-02-2018	10 125 076
	Wednesday-28-02-2018	10 387 619
CIC-IDS-2017	Friday-07-07-2017	697 441
	Monday-03-07-2017	554 097
	Thursday-06-07-2017	522 598
	Tuesday-04-07-2017	485 164
	Wednesday-05-07-2017	697 909
CIC-DoS-2017	AppDDoS	317 081

Table 5.1: The number of the exported flows with the name of the respective capture files and the corresponding dataset.

of the correlated features is also removed."

As seen in the plot, when considering all employed datasets combined, there is no common tendency for a subset of significant features that could be selected. Instead, each dataset has its own features that are more or less important for the classification result. Thus, the complete set of extracted features is used for the information exchange within the CIDS; Intuitively that makes sense, because the initial goal was to provide a general knowledge database, which each member of the CIDS can incorporate in its detection pipeline with their individual preprocessing steps for the final classification training

```
Input: fitted model $M$
Output: Feature importances $I$
Compute the reference score $s$ of the model $M$ on data $\mathcal{X}$
for each feature column $x_j$ do
    for each repitition $i$ in $1, \ldots K$ do
        randomly shuffle column $j$ of dataset $\mathcal{X}$ to generate a corrupted version of the data named $\hat{\mathcal{X}}_{\rangle,|}$
        compute the score $s_{i,j}$ of model $M$ on $\hat{\mathcal{X}}_{\rangle,|}$
    compute importance $i_j$ for feature $x_j$ $i_j = s - \frac{1}{K}\sum_i = 1^K s_i j$
return $\mathcal{R}_{\text{out}}$
```

Algorithm 11: Permutation Importance

Dataset	Class	No. Flows	Grouped/Renamed	Label
CIC-IDS-2018	**Benign**	**92 351 111**		0
	Attack	**1 623 847**		
	Bot	95 038		1
	Brute-Force-SSH	92 618		3
	Brute-Force-Web	644		4
	DDoS-LOIC-HTTP	289 328	DDoS	5
	DDoS-LOIC-UDP	19 013		
	DDoS-HOIC	1 074 379		
	DoS-GoldenEye	28 297	DoS HTTP High Vol.	6
	DoS-Slowloris	24 044	DoS HTTP Low Vol.	7
	SQL-Injection	44		9
	XSS	442		10
	DoS-Hulk	479 829		-
	DoS-SlowHTTPTest	0		-
	Brute-Force-FTP	0		-
	Infiltration	146		-
CIC-IDS-2017	**Benign**	**554 097**		0
	Attack	**120 152**		
	Bot	736		1
	Brute-Force-SSH	3 003		3
	Brute-Force-Web	295		4
	DDoS-LOIT	93 373	DDoS	5
	DoS-GoldenEye	8 100	DoS HTTP High Vol.	6
	DoS-Slowloris	10 591	DoS HTTP Low Vol.	7
	DoS-SlowHTTPTest	3 958		
	SQL-Injection	12		9
	XSS	84		10
	DoS-Hulk	158 645		-
	Brute-Force-FTP	4 003		-
	Infiltr. Cool-Disk-MAC	14		-
	Infiltr. Dropbox	5		-
	Scan	160 426		-
	Heartbleed	82		-
CIC-DoS-2017	**Benign**	**200 575**		0
	Attack	**43 284**		
	DDoS-DDoSSim	7 231	DDoS	5
	DoS-GoldenEye	1 959	DoS HTTP High Vol.	6
	DoS-Rudy	3 310	DoS HTTP Low Vol.	7
	DoS-Slowbody	7 628		
	DoS-Slowheaders	16 386		
	DoS-Slowloris	4 643		
	DoS-Slowread	2 127		
	DoS-Hulk	2 508		-

Table 5.2: After the labeling process, the exported flow samples have partitioned to the classes presented.

ID	Feature Description	ID	Feature Description	ID	Feature Description
1	Source IP address	16	Max. packet IATs (source → destination)	31	Median packet IATs (destination → source)
2	Source Port	17	Mean packet IATs (source → destination)	32	Std. Dev. packet IATs (destination → source)
3	Dest. IP address	18	Median packet IATs (source → destination)	33	Number packets (bidirectional)
4	Dest. Port	19	Std. Dev. packet IATs (source → destination)	34	Sum packet sizes (bidirectional)
5	Protocol	20	Number packets (destination → source)	35	Min. packet sizes (bidirectional)
6	UNIX Timestamp	21	Sum packet sizes (destination → source)	36	Max. packet sizes (bidirectional)
7	Number packets (source → destination)	22	Min. packet sizes (destination → source)	37	Mean packet sizes (bidirectional)
8	Sum packet sizes (source → destination)	23	Max. packet sizes (destination → source)	38	Median packet sizes (bidirectional)
9	Min. packet sizes (source → destination)	24	Mean packet sizes (destination → source)	39	Std. Dev. packet sizes (bidirectional)
10	Max. packet sizes (source → destination)	25	Median packet sizes (destination → source)	40	Sum packet IATs (bidirectional)
11	Mean packet sizes (source → destination)	26	Std. Dev. packet sizes (destination → source)	41	Min. packet IATs (bidirectional)
12	Median packet sizes (source → destination)	27	Sum packet IATs (destination → source)	42	Max. packet IATs (bidirectional)
13	Std. Dev. packet sizes (source → destination)	28	Min. packet IATs (destination → source)	43	Mean packet IATs (bidirectional)
14	Sum packet IATs (source → destination)	29	Max. packet IATs (destination → source)	44	Median packet IATs (bidirectional)
15	Min. packet IATs (source → destination)	30	Mean packet IATs (destination → source)	45	Std. Dev. packet IATs (bidirectional)

Table 5.3: After the labeling process, the exported flow samples have partitioned to the classes presented.

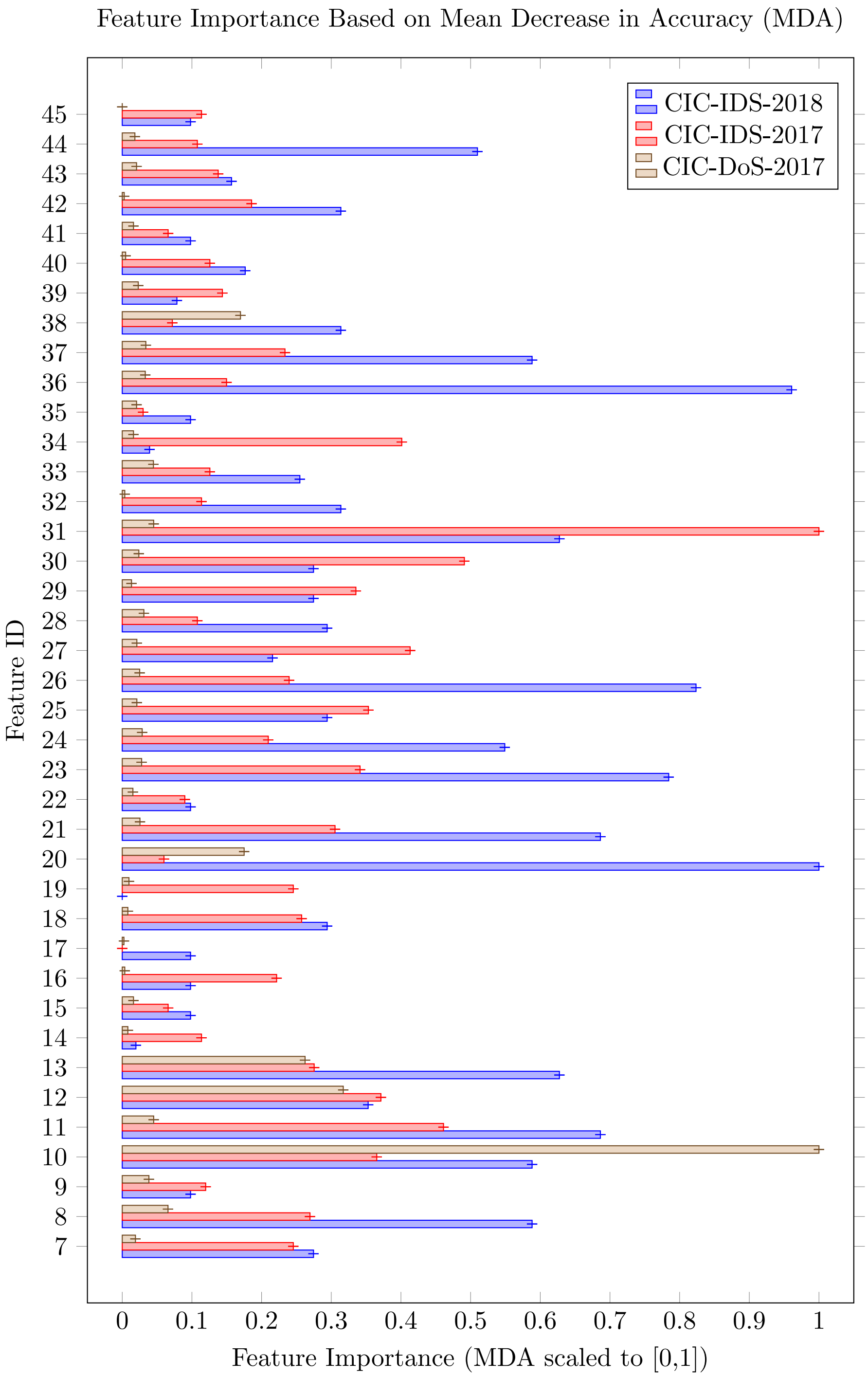


Figure 5.1: Marginals $p(x_M)$, $p(x_N)$ and Conditional Gaussian $p(x_{M|N})$

5.2 Experimental Setup

5.2.1 Implementation Details

5.2.2 Methodologies

5.3 Evaluation Results

5.3.1 Classification Improvement

5.3.2 Overhead Reduction

5.4 Summary and Discussion

Chapter 6

Conclusion and Outlook

6.1 Conclusion

6.2 Outlook

Bibliography

[Hot33] Harold Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of Educational Psychology* 24 (1933), pp. 498–520.

[Sch78] Gideon Schwarz. “Estimating the dimension of a model”. In: *The annals of statistics* (1978), pp. 461–464.

[Den87] Dorothy E Denning. “An intrusion-detection model”. In: *IEEE Transactions on software engineering* 2 (1987), pp. 222–232.

[Cla88] David Clark. “The design philosophy of the DARPA Internet protocols”. In: *Symposium proceedings on Communications architectures and protocols*. 1988, pp. 106–114.

[Seb+88] Michael M Sebring et al. “Expert systems in intrusion detection: A case study”. In: *Proceedings of the 11th National Computer Security Conference*. 1988, pp. 74–81.

[LTG92] Teresa F Lunt, Ann Tamaru, and F Gillham. *A real-time intrusion-detection expert system (IDES)*. SRI International. Computer Science Laboratory, 1992.

[PN97] Phillip A. Porras and Peter G. Neumann. “EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances”. In: *National Information Systems Security Conference* (1997).

[IM98] Piotr Indyk and Rajeev Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality”. In: *ACM Symposium on Theory of Computing* (1998).

[VK98] Giovanni Vigna and Richard A Kemmerer. “NetSTAT: A network-based intrusion detection approach”. In: *Proceedings 14th Annual Computer Security Applications Conference (Cat. No. 98EX217)*. IEEE. 1998, pp. 25–34.

[Ken99] Kristopher Kristopher Robert Kendall. “A database of computer attacks for the evaluation of intrusion detection systems”. PhD thesis. Massachusetts Institute of Technology, 1999.

[Roe+99] Martin Roesch et al. “Snort: Lightweight intrusion detection for networks.” In: *Lisa*. Vol. 99. 1. 1999, pp. 229–238.

[VK99] Giovanni Vigna and Richard A Kemmerer. “NetSTAT: A network-based intrusion detection system”. In: *Journal of computer security* 7.1 (1999), pp. 37–71.

[Axe00] Stefan Axelsson. “The base-rate fallacy and the difficulty of intrusion detection”. In: *ACM Transactions on Information and System Security (TISSEC)* 3.3 (2000), pp. 186–205.

[Ker00] Christine Keribin. “Consistent estimation of the order of mixture models”. In: *Sankhyā: The Indian Journal of Statistics, Series A* (2000), pp. 49–66.

[DW01] Hervé Debar and Andreas Wespi. “Aggregation and Correlation of Intrusion-Detection Alerts”. In: *Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2001.

[VS01] Alfonso Valdes and Keith Skinner. “Probabilistic Alert Correlation”. In: *Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2001.

[Zha+01] Zheng Zhang et al. “HIDE: a Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification”. In: *IEEE Workshop on Information Assurance and Security* (2001).

[Cha02] Moses S Charikar. “Similarity estimation techniques from rounding algorithms”. In: *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 2002, pp. 380–388.

[CM02] Frédéric Cuppens and Alexandre Miege. “Alert correlation in a cooperative intrusion detection framework”. In: *Proceedings 2002 IEEE symposium on security and privacy*. IEEE. 2002, pp. 202–215.

[DC02] Oliver Dain and Robert K. Cunningham. “Fusing A Heterogeneous Alert Stream Into Scenarios”. In: *Applications of Data Mining in Computer Security*. Springer US, 2002.

[PFV02] Phillip A. Porras, Martin W. Fong, and Alfonso Valdes. “A Mission-Impact-Based Approach to INFOSEC Alarm Correlation”. In: *Recent Advances in Intrusion Detection*. Springer Berlin Heidelberg, 2002.

[BMH03] Denny Borsboom, Gideon J. Mellenbergh, and Jaap van Heerden. “The theoretical status of latent variables.” In: *Psychological Review* (2003).

[CLF03] S. Cheung, U. Lindqvist, and M.W. Fong. “Modeling multistep cyber attacks for scenario recognition”. In: *Proceedings DARPA Information Survivability Conference and Exposition*. 2003.

[JWQ03] R. Janakiraman, M. Waldvogel, and Qi Zhang. “Indra: A Peer-to-Peer Approach to Network Intrusion Detection and Prevention”. In: *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (2003).

[JB03] E. T. Jaynes and G. Larry Bretthorst. *Probability theory: the logic of science*. Cambridge University Press, 2003.

[MI03] Patrick Miller and Atsushi Inoue. “Collaborative intrusion detection system”. In: *22nd International Conference of the North American Fuzzy Information Processing Society, NAFIPS 2003*. IEEE. 2003, pp. 519–524.

[Cla04] Benoît Claise. *Cisco Systems NetFlow Services Export Version 9*. https://rfc-editor.org/rfc/rfc3954.txt. Accessed: 2021-03-11. 2004.

[Gar+04] Joaquin Garcia et al. “Decentralized Publish-Subscribe System to Prevent Coordinated Attacks via Alert Correlation”. In: *Information and Communications Security*. Springer Berlin Heidelberg, 2004.

[WS04] Ke Wang and Salvatore J Stolfo. “Anomalous payload-based network intrusion detection”. In: *International workshop on recent advances in intrusion detection*. Springer. 2004, pp. 203–222.

[Dep+05] Ozgur Depren et al. “An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks”. In: *Expert Systems with Applications* 29.4 (2005), pp. 713–722. ISSN: 0957-4174. DOI: https:

//doi.org/10.1016/j.eswa.2005.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0957417405000989.

[Sav05] Stefan Savage. “Internet outbreaks: epidemiology and defenses”. In: *Invited Talk in the 12th Annual Network and Distributed System Security (NDSS 05)*. 2005.

[Szo05] Peter Szor. *The Art of Computer Virus Research and Defense: ART COMP VIRUS RES DEFENSE _p1*. Pearson Education, 2005.

[AI06] Alexandr Andoni and Piotr Indyk. “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions”. In: *2006 47th annual IEEE symposium on foundations of computer science (FOCS’06)*. IEEE. 2006, pp. 459–468.

[Bis06] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[MNP06] Rajeev Motwani, Assaf Naor, and Rina Panigrahi. “Lower bounds on locality sensitive hashing”. In: *Proceedings of the twenty-second annual symposium on Computational geometry*. 2006, pp. 154–157.

[ZZ06] Jiong Zhang and Mohammad Zulkernine. “A hybrid network intrusion detection technique using random forests”. In: *First International Conference on Availability, Reliability and Security (ARES’06)*. IEEE. 2006, 8–pp.

[Cla08] Benoît Claise. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information*. https://rfc-editor.org/rfc/rfc5101.txt. Accessed: 2021-03-11. 2008.

[Fun+08] Carol J. Fung et al. “Trust Management for Host-Based Collaborative Intrusion Detection”. In: *Managing Large-Scale Service Deployment* (2008).

[McK+08] Nick McKeown et al. “OpenFlow: Enabling innovation in campus networks”. In: *Computer Communication Review* (2008).

[HTF09] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009.

[ZLK09] Chenfeng Zhou, Christopher Leckie, and Shanika Karunasekera. “Decentralized multi-dimensional alert correlation for collaborative intrusion detection”. In: *Journal of Network and Computer Applications* (2009).

[MR10] Douglas C Montgomery and George C Runger. *Applied statistics and probability for engineers*. John Wiley & Sons, 2010.

[ZLK10] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. “A survey of coordinated attacks and collaborative intrusion detection”. In: *Computers & Security* 29.1 (2010), pp. 124–140.

[TB11] Brian Trammell and Elisa Boschi. “An introduction to IP flow information export (IPFIX)”. In: *IEEE Communications Magazine* 49.4 (2011), pp. 89–95.

[Mur12] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[RGH12] Damien Riquet, Gilles Grimaud, and Michaël Hauspie. “Large-scale coordinated attacks: Impact on the cloud security”. In: *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE. 2012, pp. 558–563.

[Hof+14] Rick Hofstede et al. “Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX”. In: *Communications Surveys & Tutorials, IEEE* (2014).

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. “Finding Similar Items”. In: *Mining of Massive Datasets.* 2nd ed. Cambridge University Press, 2014, pp. 68–122. DOI: 10.1017/CBO9781139924801.004.

[Lou14] Gilles Louppe. *Understanding Random Forests: From Theory to Practice.* 2014. DOI: 10.48550/ARXIV.1407.7502. URL: https://arxiv.org/abs/1407.7502.

[Shl14] Jonathon Shlens. “A Tutorial on Principal Component Analysis”. In: *arXiv:1404.1100 [cs, stat]* (2014).

[Mil+15] Aleksandar Milenkoski et al. “Evaluating computer intrusion detection systems: A survey of common practices”. In: *ACM Computing Surveys (CSUR)* 48.1 (2015), pp. 1–41.

[Sco15] David W. Scott. *Multivariate Density Estimation.* Wiley, 2015.

[Vas+15] Emmanouil Vasilomanolakis et al. “SkipMon: A locality-aware Collaborative Intrusion Detection System”. In: *IEEE International Performance Computing and Communications Conference* (2015).

[Vas16] Emmanouil Vasilomanolakis. “On Collaborative Intrusion Detection”. PhD thesis. Darmstadt: Technische Universität Darmstadt, 2016. URL: http://tubiblio.ulb.tu-darmstadt.de/82583/.

[Jaz+17] Hossein Hadian Jazi et al. “Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling”. In: *Computer Networks* 121 (2017), pp. 25–36.

[Com18] Communications Security Establishment (CSE) and Canadian Institute for Cybersecurity (CIC). *A Realistic Cyber Defense Dataset (CSE-CIC-IDS2018).* 2018. URL: https://registry.opendata.aws/cse-cic-ids2018 (visited on 02/08/2022).

[Rub18] Aviad Rubinstein. “Hardness of approximate nearest neighbor search”. In: *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing.* 2018, pp. 1260–1268.

[SHG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”. In: *4th International Conference on Information Systems Security and Privacy* (2018).

[WM18] Michael E. Whitman and Herbert J. Mattord. *Principles of information security.* Cengage Learning, 2018. ISBN: 978-1-337-10206-3.

[KG19] Rakesh Kumar and Rinkaj Goyal. “On cloud security requirements, threats, vulnerabilities and countermeasures: A survey”. In: *Computer Science Review* 33 (2019), pp. 1–48. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2019.05.002. URL: https://www.sciencedirect.com/science/article/pii/S1574013718302065.

[Ngu+19] Tri Gia Nguyen et al. “Search: A collaborative and intelligent nids architecture for sdn-based cloud iot networks”. In: *IEEE access* 7 (2019), pp. 107678–107694.

[DFO20] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning.* Cambridge University Press, 2020.

[Hin+20] Hanan Hindy et al. “A taxonomy of network threats and the effect of current datasets on intrusion detection systems”. In: *IEEE Access* 8 (2020), pp. 104650–104675.

[Bee21] Frank Beer. “A Hybrid Flow-based Intrusion Detection System Incorporating Uncertainty”. PhD thesis. University of Kassel, 2021.

[NBJ21] Parth Nagarkar, Arnab Bhattacharya, and Omid Jafari. “Exploring State-of-the-Art Nearest Neighbor (NN) Search Techniques”. In: *8th ACM IKDD CODS and 26th COMAD*. 2021, pp. 443–446.

[TFW21] A.S. Tanenbaum, N. Feamster, and D.J. Wetherall. *Computer Networks, Global Edition*. Pearson Education, 2021. ISBN: 9781292374017. URL: https://books.google.de/books?id=nT4QEAAAQBAJ.

[Pha] Peter Phaal. *sFlow Specification Version 5*. https://sflow.org/sflow_version_5.txt. Accessed: 2022-08-10.

[Wil] S. William. *Cryptography and Network Security - Principles and Practice, 7th Edition*. Pearson Education India. ISBN: 9789353942564. URL: https://books.google.de/books?id=AhDCDwAAQBAJ.

List of Abbreviations

ACC	accuracy score
GMS	generative model selection
c-ANN	c-approximate nearest-neighbour problem
cr-ANN	cr-approximate nearest-neighbour problem
BIC	bayesian information criterion
CIDS	Collaborative Intrusion Detection System
CIDSM	Collaborative Intrusion Detection System Message
DGM	Deep Generative Models
DoS	Denial of Service
DPI	Deep Packet Inspection
DR	Detection Rate
EM Algorithm	Expectation Maximization Algorithm
FPF	Flow Processing Format
FPR	False Postive Rate
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GRP	gaussian random projection
HIDS	Host-based Intrusion Detection System
IDS	Intrusion Detection System
IETF	Internet Enginnering Task Force
IPFIX	IP Flow Information Export
LSH	locality-sensitive hashing
LVM	Latent Variable Model
MLE	Maximum Likelihood Estimation
NIDS	Network-based Intrusion Detection System
NN	nearest-neighbour search problem
ONB	Orthonormal Basis
PCA	Principal Component Analysis
QoS	Quality of Service
R2L	Remote to Local
SPoF	single point of failure
U2R	User to Root

List of Symbols

General Mathematical Notation

$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
$\mathbb{R}^n$	n-dimensional vector space of real numbers
a, b, c	scalars are lowercase
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	vectors are bold lowercase
$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$	matrices are bold uppercase
$\mathcal{A}, \mathcal{B}, \mathcal{C}$	sets
$\boldsymbol{x}^\top, \boldsymbol{A}^\top$	transpose of a vector or matrix
$\boldsymbol{A}^{-1}$	inverse of a matrix
$\boldsymbol{x}^\top \boldsymbol{y}$	dot product of $\boldsymbol{x}$ and $\boldsymbol{y}$
$B = (\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3})$	(ordered) tuple
$\boldsymbol{B} = [\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}]$	matrix of column vectors stacked horizontally
$\mathcal{B} = \{\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}\}$	set of vectors
$a \in \mathcal{A}$	a is element of the set $\mathcal{A}$
$a \notin \mathcal{A}$	a is not element of the set $\mathcal{A}$
$\{a \in \mathcal{A} \mid \phi\}$	set containing all $a \in \mathcal{A}$ meeting condition ϕ
$\lvert\mathcal{A}\rvert$	cardinality of the set $\mathcal{A}$
$a \gg b$	a is much greater than b

Architecture Specification

$N \in \mathbb{N}$	number of data points; indexed by $n = 1, \dots, N$
$D \in \mathbb{N}$	number of dimensions; indexed by $d = 1, \dots, D$
$M \in \mathbb{N}$	number of members in the CIDS; indexed by $m = 1, \dots, M$
$\mathcal{X}$	input space

$\mathcal{Y}$	space of true targets, i.e., classes
$\mathcal{D}$	training dataset

List of Tables

List of Figures

List of Algorithms