

# 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

This is the abstract

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Objectives . . . . .	1
1.3	Structure . . . . .	1
<b>2</b>	<b>Preliminaries</b>	<b>3</b>
2.1	Intrusion Detection . . . . .	4
2.1.1	Intrusion Detection Systems . . . . .	4
2.1.2	Network Flow Monitoring . . . . .	6
2.1.3	Collaborative Intrusion Detection . . . . .	7
2.2	Locality Sensitive Hashing . . . . .	12
2.2.1	The Approximate Nearest Neighbour Problem . . . . .	12
2.2.2	Locality-Sensitive Hash Functions . . . . .	13
2.2.3	Random Projection . . . . .	17
2.3	Gaussian Mixture Models . . . . .	20
2.3.1	The Gaussian Distribution . . . . .	20
2.3.2	Gaussian Mixtures . . . . .	23
2.3.3	The Expectation Maximization Algorithm . . . . .	24
2.4	Principal Component Analysis . . . . .	30
2.4.1	Dimensionality Reduction . . . . .	30
2.4.2	Signal and Noise . . . . .	30
2.4.3	Formal Description . . . . .	30
<b>3</b>	<b>State of the Art</b>	<b>31</b>
3.1	Data Dissemination in Collaborative Intrusion Detection . . . . .	31
3.1.1	Requirements . . . . .	31
3.1.2	Centralized and Hierarchical CIDS . . . . .	31
3.1.3	Distributed Architectures . . . . .	31
3.2	Similarity Hashing in Malware Detection . . . . .	34
3.2.1	Motivation . . . . .	34
3.2.2	Approximate Matching . . . . .	34
3.2.3	Locality-Sensitive Hashing in IDS . . . . .	34
3.3	Generative Algorithms and Intrusion Detection . . . . .	34

<b>4</b>	<b>Generative Pattern Database</b>	<b>35</b>
4.1	High Level Overview . . . . .	36
4.1.1	Example Integration . . . . .	36
4.1.2	Clustering . . . . .	37
4.1.3	Filtering and Compression . . . . .	37
4.1.4	Global View . . . . .	38
4.1.5	Creation . . . . .	38
4.1.6	Usage . . . . .	38
4.2	System Architecture . . . . .	39
4.2.1	Pattern Database . . . . .	39
4.2.2	Event Channel . . . . .	40
4.2.3	Processing Pipeline . . . . .	40
4.3	Local Indexing . . . . .	41
4.4	Complexity Estimation . . . . .	43
4.5	Generative Fitting . . . . .	45
4.6	Classifier Fitting . . . . .	51
<b>5</b>	<b>Experimental Evaluation</b>	<b>53</b>
5.1	Flow Feature Analysis . . . . .	53
5.1.1	Datasets . . . . .	53
5.1.2	Feature Extraction and Preprocessing . . . . .	53
5.2	Experimental Setup . . . . .	53
5.2.1	Implementation Details . . . . .	53
5.2.2	Methodologies . . . . .	53
5.3	Evaluation Results . . . . .	53
5.3.1	Classification Improvement . . . . .	53
5.3.2	Overhead Reduction . . . . .	53
<b>6</b>	<b>Conclusion and Outlook</b>	<b>55</b>
6.0.1	Conclusion . . . . .	55
6.0.2	Outlook . . . . .	55
	<b>Bibliography</b>	<b>57</b>
	<b>List of Tables</b>	<b>63</b>
	<b>List of Figures</b>	<b>65</b>
	<b>List of Algorithms</b>	<b>65</b>

# 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, 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 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 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, Intrusion Detection Systems (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 advantages are significant [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).



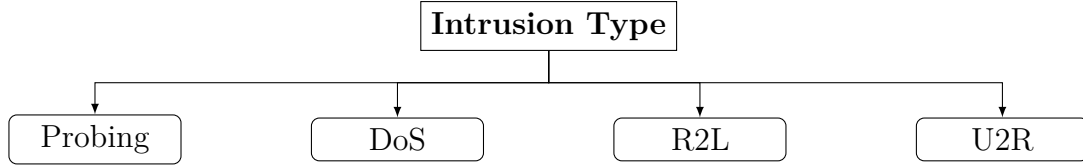


Figure 2.1: A categorization of intrusions into different subtypes

*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, IDS 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 System (HIDS) and Network-based Intrusion Detection System (NIDS). 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, IDS 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].

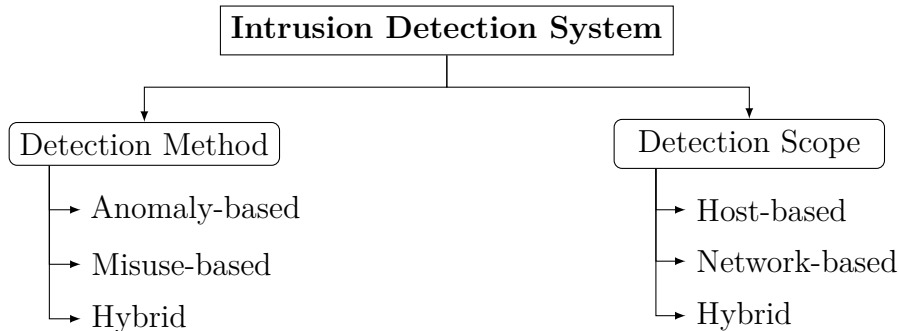


Figure 2.2: A taxonomy for IDS that distincts systems based on the utilized detection method or the scope in which it operates

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 IDS. 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 Engineering 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.

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 or more characteristics of the packet itself,
3. one or more fields derived from packet treatment.



Figure 2.3: test

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 host A to host B (or vice versa), 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 updating 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 IDS or traffic profiling.

Using network flows for intrusion detection has several advantages over 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

survey on coordinated attacks in the context of collaborative intrusion detection systems[ZLK10]

However, when considering the development of the current threat environment, the effectiveness of conventional intrusion detection systems is limited. Technology trends, such as the internet of things or cloud computing, are main drivers for increasingly

blurring corporate boundaries in the context of interconnection of infrastructures and shared resources. This transformation increases the potential attack surface of productive computer systems for large-scale and high-velocity cyber attacks, which traditional IDSs have limited effectiveness due to their isolated nature. For example, such stand-alone IDS will not be able to create connections between security events that occur at different infrastructures simultaneously. Due to the mentioned increase of attack surface that is related to the size of current computer networks, attackers may attempt to obfuscate the characteristic overall sequence of the intrusion by spreading single attack steps.

In order to address the aforementioned security problems of large IT systems, Collaborative Intrusion Detection Systems (CIDSs) have been proposed. In general, a CIDS is a network of several intrusion detection components that collect and exchange data on system security. A CIDS is essentially specified by two different types of components, namely the detection units and the correlation units, and their communication among each other. The detection units can be considered as conventional IDS that monitor a sub-network or a host and by that, generate low-level intrusion alerts. The correlation unit is 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. CIDSs pursue the following two goals [Vas16, p. 24].

- 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.
- CIDSs can monitor large-scale networks more effectively with the realization of a loadbalancing strategy. By sharing IDS resources across different infrastructures, short-term peak loads can be served, reducing the downtime of individual IDSs.

The requirements and key components are adopted from [vas\_2015], who have summarized them within an survey. It should be noted that the structuring partly overlaps and that individual or several requirements influence each other.

## Requirements

**Accuracy** 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 Positive Rate (FPR), which sets the number of normal data classified as an attack in relation to the total number of normal data.

**Minimal Overhead and Scalability** This requirement describes the operational overhead and the scalability of the system directly related to this. First, the algorithms and techniques for collecting, correlating, and aggregating data must require minimal computational overhead. Second, data distribution within the CIDS must be as 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** 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.

**Privacy** With regard to the protection and regulation of communication 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 that 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** This requirement 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.

**Interoperability** The 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.

## Design Components

**Local Monitoring** First, 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 signature-based or anomaly-based methods according to the type of detection technique used (see Chapter ??).

**Membership Management** 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 units are either centralized, hierarchical or distributed.

**Correlation and Aggregation** 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, [goo\_2001] 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, [goo\_2001b] 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 [zho\_2009].
- *Attack scenario-based* approaches detect complex attacks based on databases that provide patterns for attack scenarios (e.g. [hut\_2004], [jaj\_2002]). 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 (e.g. [cup\_2002], [che\_2003]). 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, [goo\_2002] 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** 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** 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 Neighbour Problem

For all definitions of the NN and its variants, a set of  $n$  points  $P = \{p_1, \dots, p_n\}$  in a metric space  $(X, d)$  where  $P \subset X$  is considered.<sup>1</sup> Then, the general NN is stated as follows.

**Definition 1** (Nearest Neighbour Problem). Construct a data structure so as to efficiently answer the following query: Given any query point  $q$ , find some point  $p \in P$  such that

$$\min_{q \in X} d(p, q). \quad (2.1)$$

A specific example of Definition 1 could include high-dimensional data in  $P \in \mathbb{R}^k$  with  $k \gg 1$  and define  $d$  as the euclidean distance. In this case, an exhaustive search would require a query time of  $O(kn)$ . Unfortunately, all exact algorithms that provide a better time complexity than an exhaustive search require  $O(2^k)$  space [Rub18]. 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.

**Definition 2** ( $c$ -Approximate Nearest Neighbour Problem). For any given query point  $q \in X$  and some approximation factor  $c > 1$ , find some point  $p \in P$  such that

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

---

<sup>1</sup>It is assumed that  $d$  is a proper *metric*, which means that it is *symmetric*:  $d(p, q) = d(q, p)$ , *reflexive*:  $d(p, q) \leq 0$ ,  $d(p, q) = 0 \iff p = q$  and satisfies the *triangle inequality*:  $d(p, q) \leq d(p, s) + d(s, q)$ .



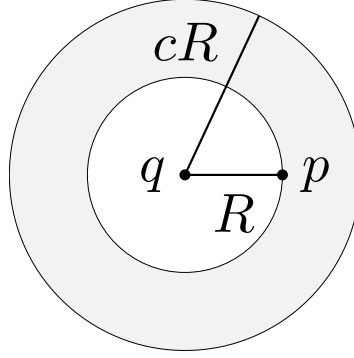


Figure 2.4: In the  $cR$ -approximate nearest neighbour problem some point within  $cR$  is accepted, if there exists a point  $p$  where  $d(p, q) \leq R$ .

Thus, the distance from the query point  $q$  to the approximate nearest neighbour  $p$  is at most  $c$  times the distance to the true nearest neighbour  $s$ . Strictly speaking, LSH does not solve the  $c$ -ANN directly. Instead, Indyk and Motwani relaxed the problem by introducing the  $cR$ -approximate nearest neighbour problem ( $cR$ -ANN) as follows.

**Definition 3** ( $cR$ -Approximate Nearest Neighbour Problem). For any given query point  $q \in X$ , some approximation factor  $c > 1$  and some target distance  $R > 0$ , if there exists a point  $p \in P$  where  $d(p, q) \leq R$ , then return a point  $p' \in P$  where

$$d(p', q) \leq cR. \quad (2.3)$$

Figure 2.4 illustrates 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. Otherwise 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 Indyk and Motwani in 1998 [IM98] as an algorithm that solves the approximate nearest neighbour problem (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.<sup>2</sup> 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<sup>3</sup>, although the most general definition applies to both.

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.

<sup>2</sup>See [NBJ21] for an exhaustive survey of NN-Search Techniques.

<sup>3</sup>In the following, cryptographic and non-cryptographic hash functions are referred to as conventional hashing.

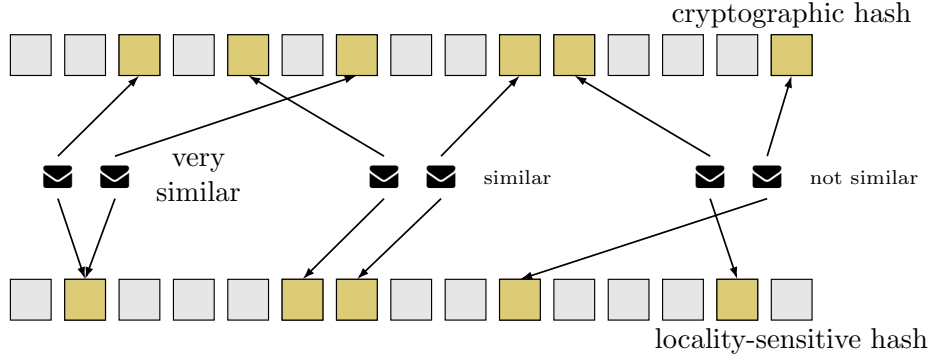


Figure 2.5: The behaviour of conventional hashing and LSH

More specifically, we define a hash function as  $h : A \rightarrow B : \mathbf{p} \mapsto \mathbf{u}$  where  $A \subset \mathbb{R}^d$  with  $d \in \mathbb{N}$  is the input set and  $B = \{0, 1\}^k$  with  $k \in \mathbb{N}$  the target set of all bit sequences of fixed size  $k$ , with  $k < d$ . According to the notation in [Cor+22, p. 256], a hash table is defined by a hash function  $h$ , that maps the keyspace  $K$  into the slots of a hash table  $T[0 \dots S - 1]$ ,  $S \in \mathbb{N}$ , i.e.  $h : K \rightarrow \{0, 1, \dots, S - 1\}$  with  $S \ll |K|$ . Thus, a message with key  $k \in K$  hashes to slot  $h(k)$ . Additionally,  $h(k)$  is the hash value of the key  $k$ .

Typically, conventional hashing is used for the realization of, e.g. hash tables, data integrity checks, error correction methods or database indexes. Depending on the application, different requirements are imposed on the utilized hash function. In this context, the most important property of a hash function is the probability of a *collision*. A collision occurs when two keys  $\mathbf{p}_1 \neq \mathbf{p}_2$  are projected onto the same hash value  $\mathbf{u} = h(\mathbf{p}_1) = h(\mathbf{p}_2)$ . For example, cryptographic hash functions are used in systems, where adversaries try to break such systems. Thus, different security requirements are defined for cryptographic hash functions [Wil, p. 349]. In particular, these hash functions are designed to be resistant against collisions, which is the key difference to LSH. 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 hash function that exhibits relaxed guarantees on the security properties in exchange for significant performance improvements. However, locality-sensitive hashes behave differently as illustrated in Figure 2.5. LSH projects similar inputs onto the same or near elements. In contrast, conventional hashing tries to distribute projections onto the elements of the target as randomly as possible.

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  $\mathcal{H} : A \rightarrow B$  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 in to the same bucket with higher probability than distant pairs.
2. Specific functions of a family need to be statistically independent, 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.

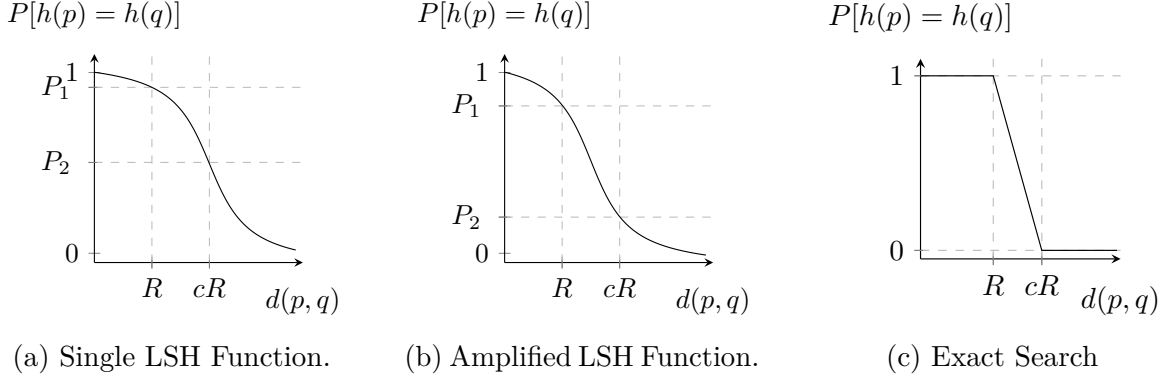


Figure 2.6: The behaviour of a  $(R, cR, P_1, P_2)$ -sensitive function in (a) and (b) (adapted from [LRU14, p. 100]) approaching the ideal probability gap in (c) resembling the behaviour of an exact search.

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

**Definition 4** (Locality-Sensitive 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  $\mathcal{H} = \{h : A \rightarrow B\}$  is called  $(R, cR, P_1, P_2)$ -sensitive if for any two points  $p, q \in A$  and any hash function  $h$  chosen uniformly at random from  $\mathcal{H}$  the following conditions are satisfied:

$$\begin{aligned} d(p, q) \leq R &\implies P[h(p) = h(q)] \geq P_1, \\ d(p, q) \geq cR &\implies P[h(p) = h(q)] \leq P_2. \end{aligned}$$

Ideally, the gap between  $P_1$  and  $P_2$  should be as big as possible as depicted in Figure 2.6c, 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 2.6a, 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 2.6b 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, sample  $K$  hash functions independently from  $\mathcal{H}$  and hash each point  $p \in P$  to a  $K$ -dimensional vector with a new constructed function  $g \in \mathcal{H}^K$  as

$$g(p) = [h_1(p), h_2(p), \dots, h_K(p)]. \quad (2.4)$$

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

$$P[h_i(p) = h_i(q)] \geq P_1 \implies P[g(p) = g(q)] \geq P_1^K \quad (2.5)$$

$$P[h_i(p) = h_i(q)] \leq P_2 \implies P[g(p) = g(q)] \leq P_2^K. \quad (2.6)$$

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  $\{g_1, \dots, g_L\}$  are constructed, where each function  $g_l$ ,  $l \in \{1, \dots, L\}$  stems from a different family  $\mathcal{H}_l$ . Note, that the algorithm is successful, when the two points  $p, q$  collide at least once for some  $g_l$ . Therefore, hashing a point  $p \in P$  with each  $g_l$  results in the following probability for collision.

$$P[\exists l, g_l(p) = g_l(q)] = 1 - P[\forall l, g_l(p) \neq g_l(q)] \quad (2.7)$$

$$= 1 - P[g_l(p) \neq g_l(q)]^L \quad (2.8)$$

$$\geq 1 - (1 - P_1^K)^L \quad (2.9)$$

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 general algorithm for solving the  $cR$ -ANN using LSH consists of two consecutive steps. First, a set of points is required for the construction of the data structure as described in Algorithm 1. For each function  $g_l$  a corresponding hash table  $T_l$  is initialized that will store all data points  $p_n$ . 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 separate chaining for example, is applied if a certain slot is already occupied.

---

**Algorithm 1** LSH Preprocessing

---

**Input:**  $N$  points  $p_n \in P$  with  $n \in \{1, \dots, N\}$ ,  $N \in \mathbb{N}$

**Output:** data structure  $\{T_1, \dots, T_L\}$

- 1: construct hash functions  $g_1, \dots, g_L$  each of length  $K$  (see Equation 2.4)
  - 2: **for each**  $g_l \in \{g_1, \dots, g_L\}$  **do**
  - 3:      $T_l \leftarrow$  new HashTable
  - 4:     **for each**  $p_n \in P$  **do**
  - 5:         add  $p_n$  to  $T_l[g_l(p_n)]$
- 

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

**Algorithm 2** LSH Query**Input:** a query point  $q$  and data structure  $\{T_1, \dots, T_L\}$ **Output:** a set  $S$  of nearest neighbours of  $q$ 


---

```

1:  $S \leftarrow \emptyset$ 
2: for each  $g_l \in \{g_1, \dots, g_L\}$  do
3:    $P \leftarrow T_l[g_l(q)]$ 
4:   if  $|P| > 0$  then
5:     for each  $p_n \in P$  do
6:       if  $d(p_n, q) \leq cR$  then
7:         add  $p_n$  to  $S$ 
8: return  $S$ 

```

---

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], resulting in the following theorem.

**Theorem 5.** Let  $(X, d)$  be a metric on a subset of  $\mathbb{R}^d$ . Given a  $(R, cR, P_1, P_2)$ -locality sensitive hash family  $\mathcal{H}$  and write  $\rho = \frac{\log(1/P_1)}{\log(1/P_2)}$ . Then for  $n = |X|$  and for any  $n \geq \frac{1}{P_2}$  there exists a solution to the  $cR$ -ANN with space complexity  $O(dn + n^{1+\rho})$  and query time of  $O(n^\rho)$ .

Therefore, it follows that the query time of LSH can only be sublinear, if  $\rho < 1$ , which is the case if the inequality  $P_1 > P_2$  is satisfied.

### 2.2.3 Random Projection

This section presents a specific family  $\mathcal{H}$  that is locality-sensitive. The algorithm that bases on this family 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 GRP is a form of LSH.

The cosine distance can be applied in euclidean spaces and discrete versions of euclidean spaces [LRU14, p. 95]. For two real vectors  $p_1$  and  $p_2$ , the cosine distance between is equal to the the angle between  $p_1$  and  $p_2$ , 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  $p_1$  and  $p_2$ , the cosine distance  $\theta(p_1, p_2)$  is the dot product of  $p_1$  and  $p_2$  divided by their euclidean distances from the origin ( $L_2$ -norm):

$$\theta(\mathbf{p}_1, \mathbf{p}_2) = \cos^{-1} \left( \frac{\mathbf{p}_1 \cdot \mathbf{p}_2}{\|\mathbf{p}_1\| \|\mathbf{p}_2\|} \right). \quad (2.10)$$

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

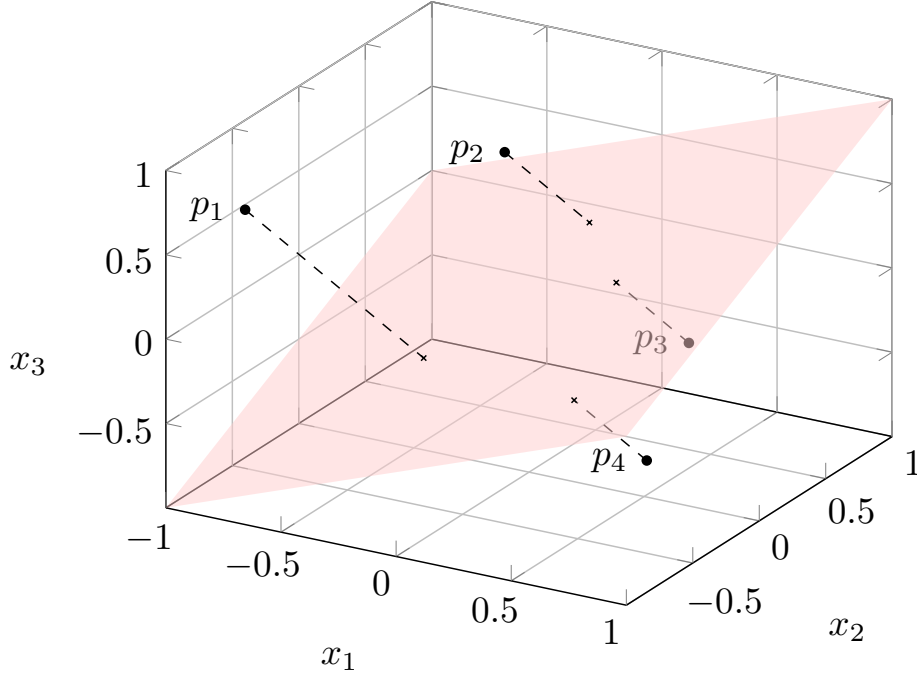


Figure 2.7: Illustration of a random hyperplane (red) partitioning the space

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

$$1 - \frac{\theta(\mathbf{p}_1, \mathbf{p}_2)}{\pi} \quad (2.11)$$

Introduced in [Cha02], the GRP is defined as follows. Given a message  $\mathbf{x} \in A = \{x \in \mathbb{R}^D : 0 \leq x \leq 1\}$  and a randomly selected hyperplane defined as  $\mathbf{M} = (a_{ij}) \in \mathbb{R}^{D \times K}$  where  $a \sim \mathcal{N}(0, I)$ , a *gaussian random projection (GRP)* aims to (I) reduce the dimensionality from  $D$  to  $L$  dimensions and (II) provide a binary encoding by first projecting  $\mathbf{x}$  onto  $\mathbf{M}$  and subsequently applying the sign function to each element of the result, which can be formalized as

$$h(\mathbf{x}) = [h(\mathbf{x}, a_1), \dots, h(\mathbf{x}, a_k)] \text{ with } h(\mathbf{x}, a) = \text{sign}(\mathbf{x}^T a) \quad (2.12)$$

$$\text{with } \text{sign}(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases} \quad (2.13)$$

An illustration of a random hyperplane that dissects the three-dimensional space and partitions the data space is given in Figure 2.7. The resulting digest  $h(\mathbf{x})$  is a binary vector  $h(\mathbf{x}) = \mathbf{u} \in B = \{0, 1\}^l$  that is used as bucket index for storing  $\mathbf{x}$  in a hash table. For any two messages  $\mathbf{x}_1, \mathbf{x}_2$ , the probability of being hashed to the same bucket increases with a decreasing distance, given as

$$P[h(\mathbf{x}_1) = h(\mathbf{x}_2)] = 1 - \frac{\theta(\mathbf{x}_1, \mathbf{x}_2)}{\pi}. \quad (2.14)$$

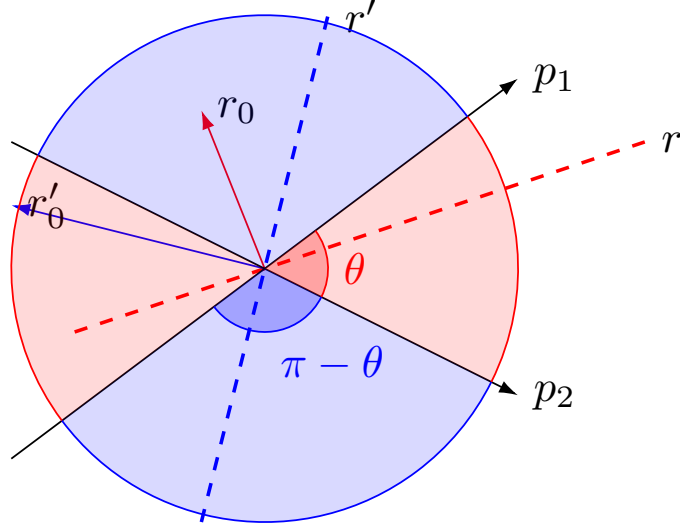


Figure 2.8: Visual Proof of claim in equation 2.14

For a visual proof of the claim in Equation 2.14 consider Figure 2.8, where two vectors  $p_1$  and  $p_2$ , regardless of their dimensionality, define a plane and an angle  $\theta$  in this plane. Pick a hyperplane  $r$  that intersects the plane that is spanned by  $p_1$  and  $p_2$  in a line (depicted as a red dashed line). Actually, a normal vector  $r_0$  is randomly selected and  $r$  is the set of points whose dot product with  $r_0$  is zero. Since  $p_1$  and  $p_2$  are on different sides of the hyperplane, their projections given by  $\langle p_1, r_0 \rangle$  and  $\langle p_2, r_0 \rangle$  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  $r'_0$  that is normal to the hyperplane  $r'$ , which is represented by the blue dashed line. Both  $\langle p_1, r'_0 \rangle$  and  $\langle p_2, r'_0 \rangle$  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  $p_1$  and  $p_2$  are equally likely. Thus, the probability that the hyperplane looks like the red line is  $\theta/\pi$  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 refers to the selection of a number of  $K$  different random hyperplanes  $r_k$ . According to the construction in Equation 2.12, all projections  $\langle p, r_k \rangle$  for a single point are concatenated into 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 Mixture Models

This section focuses the concept of GMM for density estimation. For this purpose, the Gaussian normal distribution is presented first, by introducing basic definitions for the probability density functions and describing the closure properties of the distribution. For this purpose, at the beginning a short motivation for the application of the presented model will be given. Then, mixture models in general and GMM in particular are defined. After that, two main concepts will be explained. First, the GMM is interpreted in terms of a LVM. Second, the 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  $\mathbf{X} = (X_1, \dots, X_D)^T$ , where each element  $X_d$  with  $d \in \{1, \dots, D\}$ ,  $D \in \mathbb{N}$  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

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

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

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

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

with  $\mathbf{x} \in \mathbb{R}^D$ . The mean vector specifies the estimate of the expected value of the distribution, with each of its components describing the mean  $\mu$  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.



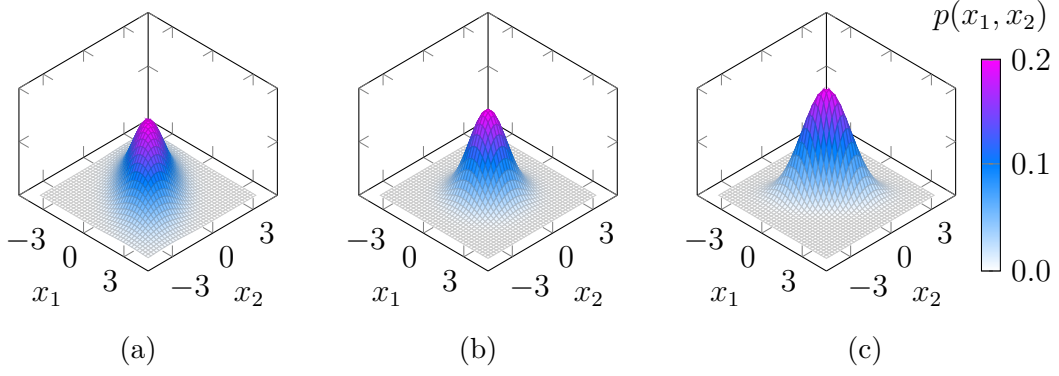


Figure 2.9: Bivariate gaussian distributions exhibit different shapes with a changing correlation value between the random variables  $x_1$  and  $x_2$ : (a) negative, (b) zero and (c) positive correlation

An illustration of the influence of the described parameters on the location and shape of a multivariate distribution is given in Figure 2.9. 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  $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , we partition  $\mathbf{X}$  according to

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_M \\ \mathbf{X}_N \end{bmatrix},$$

with  $\mathbf{X}_M \in \mathbf{R}^M$ , where  $M < N$ , and  $\mathbf{X}_N \in \mathbf{R}^N$ , where  $N = D - M$ . In general,  $\mathbf{X}_M$  is chosen to be the first  $M$  elements of  $\mathbf{X}$  and  $\mathbf{X}_N$  the rest. Accordingly,  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  are partitioned as

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

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  $\mathbf{X} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , with partitions  $\mathbf{X}_M, \mathbf{X}_N$ , the distributions of  $\mathbf{X}_M$  and  $\mathbf{X}_N$  are called marginals and their corresponding probability density function can be obtained by

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

Furthermore, each partition  $\mathbf{X}_M, \mathbf{X}_N$  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

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

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  $\mathbf{X} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , with partitions  $\mathbf{X}_M, \mathbf{X}_N$ , the conditional distribution is defined as

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

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

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

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 2.10. 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  $p(x_M, x_N)$ .

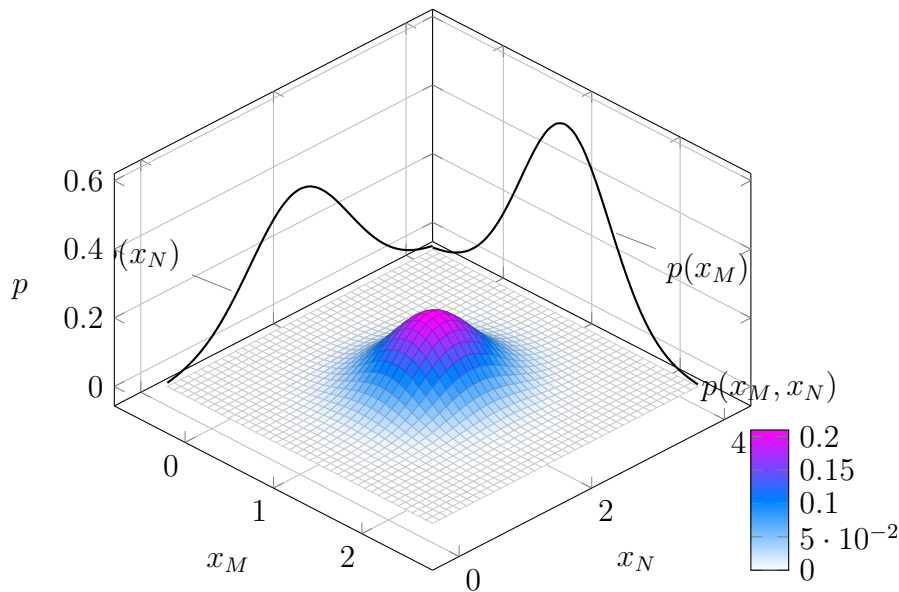


Figure 2.10: Marginals  $p(x_M)$ ,  $p(x_N)$  and Conditional Gaussian  $p(x_{M|N})$

## 2.3.2 Gaussian Mixtures

In this section, the GMM is presented as a model for density estimation. For this purpose, at the beginning a short motivation for the application of the presented model will be given. Then, mixture models in general and GMM in particular are defined. After that, two main concepts will be explained. First, the GMM is interpreted in terms of a LVM. Second, the EM Algorithm is presented, which performs the computation of model parameters of latent variable models in an iterative scheme.

### Motivation

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, looking at the marginal distribution  $p(x_1)$  in Figure ??, it is apparent that the data follow a multimodal distribution, i.e., have more than one center. A density estimate using a simple Gaussian distribution is not sufficient to effectively represent such multimodal data. Therefore, a more flexible type of distribution needs to be introduced that can be used for density estimation.

### Definition

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  $\pi_k$ , 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 ?. Furthermore, the quality of the model, in terms of its ability to estimate real data distributions, is theoretically supported by the Central Limit Theorem, which states, that most data distributions converge to a normal distribution on average as the number of data points increases (see [jay\_2003] for a proof).

**Theorem 15** (Central Limit Theorem). [montgomery\_2010] Consider  $N$  i.i.d. random variables  $X_i$  with  $E[X_i] = \bar{x}$  and  $\text{var}[X_i] = \sigma^2$  and let  $S_N = \sum_{i=1}^N X_i$ . It can be

shown that, as  $N$  increases, that

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

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

$$\begin{aligned} p(\mathbf{x}|\theta) &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\theta_k), \\ 0 \leq \pi_k \leq 1, \sum_{k=1}^K \pi_k &= 1, \\ \theta &:= \{\bar{\mathbf{x}}_k, \mathbf{C}_k, \pi_k : k = 1, \dots, K\}. \end{aligned} \tag{2.15}$$

### 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 [bor\_2003]. 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 [mur\_2012].

For simplicity, first a single data point  $\mathbf{x}$  is considered, which is later expanded to a set of data points  $\mathbf{X} := \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ . It is further assumed a GMM with  $K$  components. Thus, a  $K$ -dimensional binary random variable  $\mathbf{z}$  is introduced, that indicates whether the  $k$ th mixture component is responsible for generating the data point  $\mathbf{x}$ . Hence, only a particular element  $z_k$  is equal to one and all other elements are equal to zero, such that

$$z_k \in \{0, 1\}, \quad \sum_{k=1}^K z_k = 1. \tag{2.16}$$

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

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z})p(\mathbf{x}|\mathbf{z}). \quad (2.17)$$

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

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\bar{\mathbf{x}}_k, \mathbf{C}_k) \quad \Rightarrow \quad p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}|\bar{\mathbf{x}}_k, \mathbf{C}_k)^{z_k}. \quad (2.18)$$

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

$$p(\mathbf{z}) = \boldsymbol{\pi} = [\pi_1, \dots, \pi_K]^T = \prod_{k=1}^K \pi_k^{z_k}, \quad \sum_{k=1}^K \pi_k = 1. \quad (2.19)$$

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

$$p(z_k = 1) = \pi_k. \quad (2.20)$$

Then the conditional distribution  $p(\mathbf{x}|\mathbf{z})$  from (2.18) and the prior distribution  $p(\mathbf{z})$  from (2.19) are plugged into the joint distribution  $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$  from (2.17) and the marginal distribution  $p(\mathbf{x})$  is obtained by summing the joint distribution over all possible states of  $\mathbf{z}$ , which results in

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\bar{\mathbf{x}}, \mathbf{C}). \quad (2.21)$$

It can be confirmed that the GMM has been extended by including the discrete latent variable  $\mathbf{z}$  and that the result is still consistent with the previous definition, since the marginal distribution  $p(\mathbf{x})$  is a Gaussian mixture of the form (2.15). This enables the joint distribution  $p(\mathbf{x}, \mathbf{z})$  to be used in place of the marginal distribution  $p(\mathbf{x})$ , greatly simplifying parameter estimation by applying the EM Algorithm.

So far, it was assumed that the dataset consists of a single data point  $\mathbf{x}$ . Now, the dataset is extended to  $N$  data points that can be denoted by the matrix  $\mathbf{X} \in \mathbb{R}^{N \times D}$ . From that, it follows that every data point  $\mathbf{x}_n$  possesses its own latent variable  $\mathbf{z}_n$ . Similarly, all latent variables can be summarized by the matrix  $\mathbf{Z} \in \mathbb{R}^{N \times K}$ . This extension to a full dataset is also illustrated comparatively in Figure ??.

In accordance to the full dataset, the corresponding posterior probability after observing  $\mathbf{x}$ , referred to as  $r_{nk}$ , is obtained using Bayes' Theorem as

$$\begin{aligned}
r_{nk} = p(z_{nk} = 1 | \mathbf{x}_n) &= \frac{p(z_{nk} = 1)p(\mathbf{x} | z_k = 1)}{p(\mathbf{x})} \\
&= \frac{p(\mathbf{x}_n) | z_{nk} = 1 p(z_{nk} = 1)}{\sum_{j=1}^K \pi_j p(\mathbf{x}_n | z_{nj} = 1) p(z_{nj} = 1)} \\
&= \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \bar{\mathbf{x}}_k, \mathbf{C}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \bar{\mathbf{x}}_j, \mathbf{C}_j)}.
\end{aligned} \tag{2.22}$$

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  $\mathbf{x}_n$ .

### Maximum Likelihood Estimates

The central question is how to fit the set of unknown parameters  $\theta$  to a given set of data  $\mathbf{X}$  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  $\mathbf{x} \in \mathbf{X}$  given a specific parametrization  $\theta$  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

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

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

The common solution scheme that would follow would be to calculate the gradient  $\partial \mathcal{L} / \partial \theta$ , set it to zero and solve for  $\theta$ . 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:

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

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].

$$\begin{aligned} \bar{\mathbf{x}}'_k &= \frac{\sum_{n=1}^N r_{nk} \mathbf{x}_n}{\sum_{n=1}^N r_{nk}} \\ \mathbf{C}'_k &= \frac{1}{N_k} \sum_{n=1}^N r_{nk} (\mathbf{x}_n - \bar{\mathbf{x}}_k)(\mathbf{x}_n - \bar{\mathbf{x}}_k)^T \\ \pi'_k &= \frac{N_k}{N} \end{aligned} \tag{2.23}$$

### Expectation Maximization Algorithm

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 (2.22) is calculated for all data points  $\mathbf{x}_n$  with the current set of parameters  $\theta^{(t)}$ . 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 (2.23). At the end of each iteration, the log likelihood for the new set of parameters  $\theta^{(t+1)}$  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  $\epsilon$  and the algorithm stops if no significant change occur anymore.

Figure ?? illustrates the EM algorithm for fitting a GMM with three components to the two-dimensional dataset from Figure ?. The resulting GMM with its component parameters can also be seen in example ?. Each component is represented by an coloured ellipse, that is formed by its mean and covariance matrix. Every single data point  $\mathbf{x}_n$  is associated with an probability vector, that describes each component's responsibility for creating  $\mathbf{x}_n$ . This is visualized by coloring each data point according to its probability vector, i.e. a mixed proportion of purple, cyan and orange. At the beginning, the three component parameters are chosen randomly and no posterior  $r_{nk}$  is computed yet and thus no data point is colored. With each iteration, the parameters approach the optimum, which they reach after the 21st iteration, and the algorithm converges.

How many components should be chosen for fitting a GMM? A common method is to select the model  $M$  with the highest probability given the data  $D$ . Assuming that the model  $M$  is completely described by its set of parameters  $\theta$ , the maximum likelihood function of the model  $M$  is given by

$$\hat{L} = p(D|\hat{\theta}, M), \quad (2.24)$$

answers the question of what is the probability that  $D$  is explained by  $M$ . Note, that the carat denotes the parameters that maximize the probability, i.e., the maximum likelihood function. Theoretically, in the case of a *GMM*, one could just increase the number of parameters  $K$ , e.g., the number of components, arbitrarily until the  $K = N$  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  $L$  has to be balanced against the number of model parameters  $K$ . The Bayesian information criterion (BIC) is considered as a standard criterion for model selection of GMM because of its theoretical consistency in choosing the number of components [Ker00].

In general, the BIC can be defined as

$$\text{BIC} = K \ln(N) - 2 \ln(\hat{L}), \quad (2.25)$$

which derives from the findings in [Sch78]. The BIC balances the number of model parameters  $K$  and number of data points  $N$  against the maximum likelihood function



$L$ . In the model selection, the optimal number of model parameters  $K$  minimizes the BIC, such that the BIC provides a principled way of selecting between multiple different models. More complex models almost always fit the data better, resulting in a lower value of  $-2\ln(\hat{L})$ . The BIC penalizes extra parameters by introducing the term  $K\ln(N)$ . Beyond penalizing more parameters, it furthermore assists in making a judgement as to how the additional parameters improve the model in the presence of more data.

Considering a mixture model with  $K$  components defined by

$$\begin{aligned} p(\mathbf{x}|\theta) &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\theta_k), \\ 0 &\leq \pi_k \leq 1, \sum_{k=1}^K \pi_k = 1, \\ \theta &:= \{\bar{\mathbf{x}}_k, \mathbf{C}_k, \pi_k : k = 1, \dots, K\}, \end{aligned} \tag{2.26}$$

the parameters of  $K$  multivariate normal distributions are to learn, with  $d$  dimensions, 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. Additionally,  $K$  mixture weights have to be determined. Since these sum to one, it is sufficient to determine  $K-1$  weights, leading to  $Kd + K(d(d+1)/2) + K-1$  parameters. Thus, the BIC for a dataset  $D$  with  $N$  datapoints of dimensionality  $d$  and a GMM  $M$  as defined above is stated as

$$\text{BIC}(M|D) = (Kd + K(d(d+1)/2) + K-1) \ln(N) - 2\ln(\hat{L}). \tag{2.27}$$

A model selection algorithm could look like the following.

---

**Algorithm 3** GMM Selection with BIC

---

**Input:** Dataset  $D$

**Output:** Gaussian Mixture Model  $M$

```

1:  $B \leftarrow$  new Array
2:  $\text{GMM} \leftarrow$  new Array
3: for each  $k$  in  $\{1, \dots, K\}$  do
4:    $M_k \leftarrow \text{fitGMM}(D)$ 
5:    $b_k \leftarrow \text{BIC}(M_k|D)$ 
6:    $\text{GMM}[k] \leftarrow M_k$ 
7:    $B[k] \leftarrow b_k$ 
8:  $\hat{k} \leftarrow \text{argmin}(B)$ 
9:  $\hat{M} \leftarrow \text{GMM}[\hat{k}]$ 
10: return  $\hat{M}$ 
```

---

## **2.4 Principal Component Analysis**

### **2.4.1 Dimensionality Reduction**

### **2.4.2 Signal and Noise**

### **2.4.3 Formal Description**

# Chapter 3

## State of the Art

### 3.1 Data Dissemination in Collaborative Intrusion Detection

#### 3.1.1 Requirements

#### 3.1.2 Centralized and Hierarchical CIDS

#### 3.1.3 Distributed Architectures

The key components for designing a CIDS architecture are described in [Vas16, p. 34]. Among them, data dissemination is particularly noteworthy as one of the fundamental components for the communication between members. A central aspect is the communication *overhead* that is introduced with the dissemination of alerts and knowledge within a CIDS, which in turn is heavily influenced by the CIDS architecture [Vas16, p.39], that can be primarily categorized into centralized, hierarchical and decentralized approaches [ZLK10] (see Figure ??). Centralized architectures, consisting of multiple monitoring units and a central analysis unit [CM02][MI03], suffer from a Single Point of Failure (SPoF) and are limited in their scalability due to a bottleneck, which is introduced by the central analysis unit. However, the SPoF problem can be mitigated if individual components in such a system 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, which can also logically be regarded as a central data repository. 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]. Again, such disadvantages only play a major role in non-redundantly implemented systems. Additionally, there exist approaches that are inherently distributed. Distributed architectures [Vas+15; Gil+13; BA08; Fun+08; JWQ03], wherein each participating system has a monitoring and analysis feature 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 flow of information in centralized and hierarchical architectures is governed by the logical arrangement of components and their specific role. In centralized architectures, there is usually a central entity that controls communication. In hierarchical systems, the communication paths are mainly governed by the topological structure. In contrast to that, different techniques exist in distributed approaches. Whereas flooding refers to unfiltered data distribution to all members, selective techniques reduce the overhead by using, e.g., random walks [VF06], gossiping approaches [Das+06][GKM03] or publish-subscribe mechanisms. The latter provide flexible organization options and guarantee delivery. For example, subscriptions utilized in [JWQ03] are based on special attack forms.

The data dissemination strategy of a CIDS not only defines the communication paths as described above, but also influences what kind of data is exchanged between the CIDS members while also specifying format and level of granularity. This also includes the central aspects of *data privacy*, realized by utilization of, e.g., bloom filters [Vas+15][Loc+05], and *interoperability* that can be obtained by standardized formats, such as the Intrusion Detection Message Exchange Format [CM02][Dum+06].

In particular, there are two approaches to mention, that directly address the problems of communication overhead and privacy in this context. In [Loc+05], the authors present an approach for the efficient and privacy aware distribution of alert data in a distributed CIDS. Before exchanging information, the respective data is compressed by the utilization of a bloom filter data structure. Upon an alert, the relevant information, e.g. IP address, is inserted into the bloom filter. Since the bloom filter contains information on suspicious hosts, it is called *watchlist*. The watchlist is shared among peers, which are selected by a network scheduling algorithm called *whirlpool*, that dynamically creates an overlay that defines peer relationships.

The authors state, that within data dissemination, there are two challenges. First, the disclosure of sensitive data, e.g. IP addresses, to collaborative entities renders participation in the collaboration not an option. Second, the tradeoff between latency and accuracy of the information exchange and the required bandwidth. Centralized approaches disseminate information reliably and predictably, but they constitute a bottleneck. While distributed approaches scale well, information can be lost or delayed by partitioning the data among the peers.

In the context of reducing communication overhead, this approach addresses this challenge twofold. First, alert data is compressed when inserted into the bloom filter by hashing. Second, only peers exchange data, which reduces overhead significantly, when compared to a complete distribution. However, bloom filters are probabilistic data structures, which exhibit increasing false positive matches with increasing filling degree. Thus, when scaling this approach, the probability that innocent hosts are considered as malicious, increases.

Also, the authors in [Vas+15] state, that a CIDS needs to provide *scalability*, minimal message *overhead*, *privacy* of the exchanged alert data, an *domain awareness*, which describes the ability to constrain alert dissemination to specific sub-domains of a network. Instead of randomly creating sub-domains as in [Loc+05], the authors suggest to incorporate network traffic similarities into account for this process.

data dissemination: extract specific features from alerts and add data into bloom filter;

then utilize data dissemination technique (e.g. flooding, partial flooding, gossiping) for sending bloom filters to other nodes (peers)

similarity (alert) correlation: when nodes receive data from other nodes, they compute their similarity value by performing logical operations on the bloom filters

similarity (alert) correlation: when nodes receive data from other nodes, they compute their similarity value by performing logical operations on the bloom filters, after calculating the similarity value, nodes will make use of a threshold value  $t$  to determine whether the similarity value is enough for joining a group (community creation)

Community formation: After the successful dissemination and correlation of the alert data, each sensor creates a matrix with its local knowledge of other sensors. Based on this knowledge and along with the utilized threshold, sensors can identify others and form a community with them to, afterwards, exchange more fine-grained alert data.

The problem of finding an optimal threshold value (golden standard) heavily depends on the network that is to be monitored.

To sum up, the following challenges in the context of data dissemination are found to be critical for the success of the overall system.

**Minimal Overhead** Detection latency can be affected by various factors, some of which are encountered in conventional IDS and others that are specifically relevant in the CIDS context. With regards to the data dissemination in CIDS, the computational overhead introduced by the communication of multiple monitors in the CIDS needs to be minimized, such that potential knowledge gains are available fast enough.

**Privacy** Members may not want to disclose data that contains information on system- and network states of their infrastructure, as it constitutes a privacy and security problem. This includes, among other things, legal aspects when it comes to sharing log and network data. Nonetheless, the exchange of this information is crucial for the effective operation of a CIDS.

**Interoperability** The 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.

**Domain Awareness** t.b.d. (a feature that increases scalability by reducing overhead and contributes to privacy by partially constraining communication; also increases accuracy, because only those alerts are shared that are relevant for w.r.t. to similarity of data etc.)

## 3.2 Similarity Hashing in Malware Detection

### 3.2.1 Motivation

### 3.2.2 Approximate Matching

### 3.2.3 Locality-Sensitive Hashing in IDS

Searching for similar objects in large data sets is a fundamental challenge that has found important applications in many fields. An important subclass of similarity search is the nearest neighbour search problem, which becomes hard in the high dimensional case, when relying on exact algorithms like a linear scan ( $O(n)$ ) as the best solution. However, many applications do not require an exact solution, such that in these cases a randomized algorithm can be used, which provides the correct solution with high probability in sublinear time [Dat+04]. Therefore, approximate and randomized solution algorithms are widely used in practice. There is popular approach known as locality-sensitive hashing (LSH) that allows searching in large databases for similar items in a randomized manner. This technique hashes similar input onto the same hash code with high probability for the purpose of populating a hash table. Since similar items are placed into the same buckets, this approach is suitable for data clustering or nearest neighbour search.

Also the field of cyber security has adopted methods suitable for similarity search, mainly for the analysis of polymorphic malware. This type of malware poses a significant challenge, since it changes its appearance over time in order to stay undetectable from antivirus software [WM18, p.91]. Traditional antivirus software uses cryptographic hash functions (SHA-256) to create file signatures. Such signatures are well suited to search for identical files in a knowledge database. However, due to the property of cryptographic diffusion, even minimal changes to the malware result in large differences in the resulting hash value. With the rapid evolution and proliferation of polymorphic malware, detection based on unique signatures no longer seems effective.

Based on these developments, detection schemes based on approximate matching algorithms have initially become the focus of research. In contrast to LSH, approximate matching functions are designed for producing digests of objects and a subsequent comparison of such digests, which yields a confidence value reflecting the similarity of two objects [MH17]. Popular approaches in this area are for example *ssdeep*, which implements context triggered piecewise hashing (CTPH) [Kor06] or *sdhash*, that makes use of bloom filters for the digest creation and comparison [Rou10].

## 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.

While Section 4.2 gives a high-level overview of the proposed architecture and its main processing primitives, details on the specific algorithms and strategies of the main

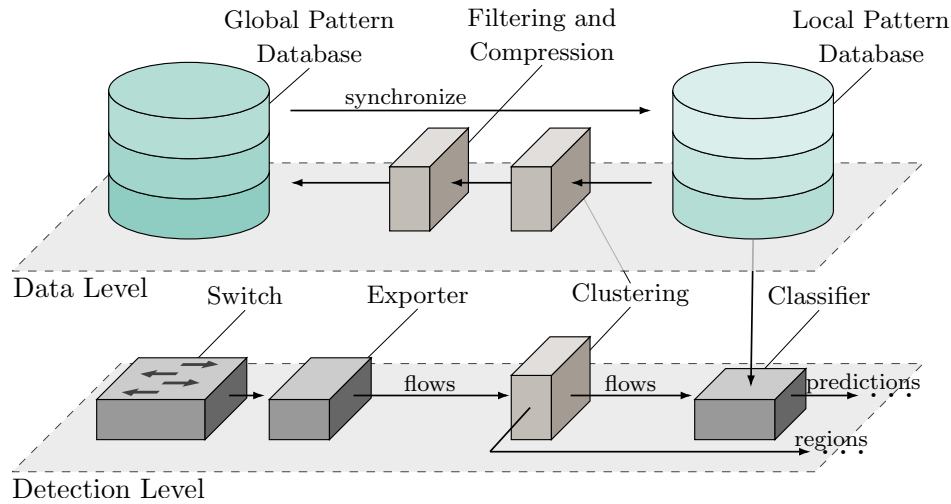


Figure 4.1: Integration of the approach into a generic NIDS.

services are described in Sections 4.3 to 4.6.

## 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 usage 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 shown NIDS consists of three components, which 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 for that member, 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, is subject to a set of clustering,



filtering and compression operations. This way, data privacy is maintained and the overhead is minimized by reducing the data volume. On the receiving side, the global pattern database combines data from all members to 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 stealthy attacks, which are executed on the resources of multiple CIDS members simultaneously.

### 4.1.2 Clustering

Since the results of the clustering operation should be consistent, while its execution is distributed among all members of the CIDS, an unsupervised algorithm with few parameters for initialization has to be selected. 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, random projection is a good choice.

By using random projection, 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 random projections 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. 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

### 4.1.5 Creation

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 its respective dataset. 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. If a region contains more than one class, it potentially exhibits a non-linear decision boundary, hence it is called complex. Otherwise, a region is called simple. Attack data within a complex region, separated by its label, is used as training data for a generative algorithm. Subsequently, the resulting model is transferred to the global view.

### 4.1.6 Usage

A synchronization process disseminates the region complexity estimations and the generative models to all local views. By sampling data from multiple generative models, a synthetic dataset is assembled, which is blended into the respective local datasets for enhancing the subsequent training of a discriminative model.

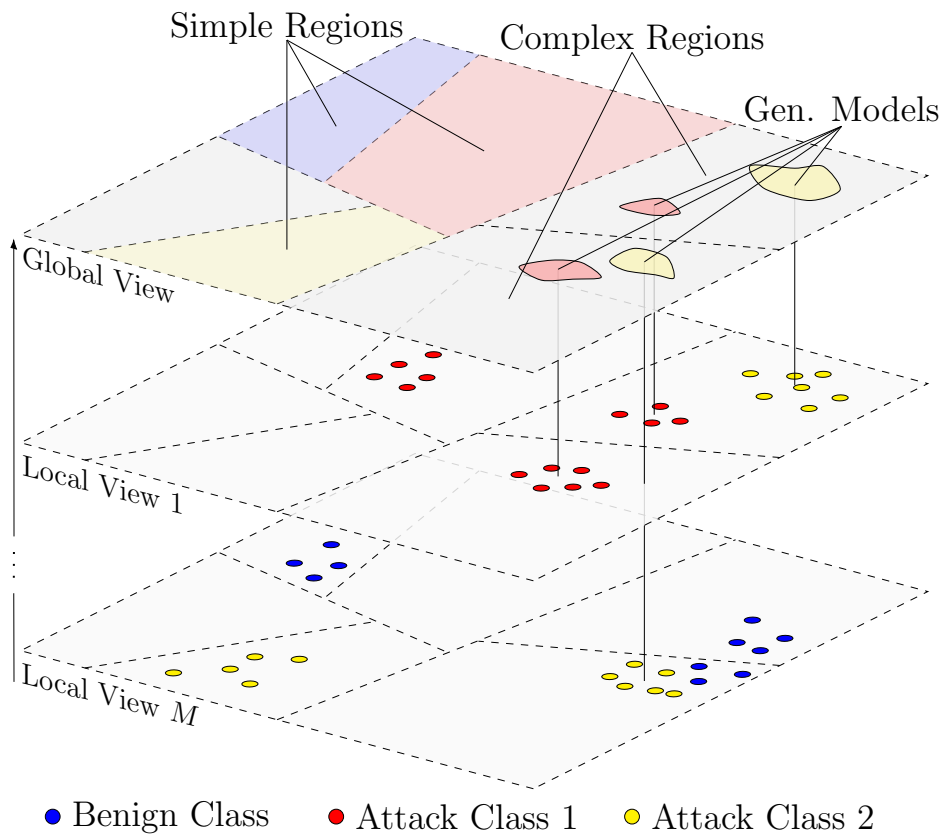


Figure 4.2: Building the global view by combining  $M$  local views.



key  $k$  in the global pattern database. Note that if a specific hash function is already used to construct a key (e.g. Random Projection), the hash table will internally apply a distinct hash function on the key to ensure even distribution across the slots.

### 4.2.2 Event Channel

Event channels provide a topic-based publish-subscribe messaging mechanism that is mainly used to 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 pattern database 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.

### 4.2.3 Processing Pipeline

Local and global pattern databases serve exclusively as data sources and sinks for operations. The only exception is the initial import of datasets  $D_m$  into the *Local Indexing* service via the messaging system.

Notation	Description
$G$	Global Infrastructure
$L_m$	Local Infrastructure $m$
$PDB_G, PDB_{L_m}$	Global Pattern Database, Local Pattern Database of $L_m$
$C_G, C_{L_m}$	Global Event Channel, Local Event Channel of $L_m$
$M \in \mathbb{N}$	Total number of CIDS participants
$m \in \{1, \dots, M\}$	Local Infrastructure Identifier

Table 4.1: Summary of the architecture notation.

### 4.3 Local Indexing

The local indexing service is responsible for the preprocessing and local storage of intrusion detection datasets. 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 locality-sensitive hash function. Based on the generated hash value, a key is constructed that is used to persist the data point in the respective local pattern database. 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 formed 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.

---

**Algorithm 4** Preprocessing and inserting  $B \subset D_m$  into  $PDB_{L_m}$

---

**Input:** Pairs of datapoints and labels  $B \leftarrow [(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_b, y_b)]$

**Output:** Regions  $R$

```

1:  $R \leftarrow$  new Set
2: for each  $(\mathbf{x}, y)$  in  $B$  do
3:    $\mathbf{x}' \leftarrow \text{normalize}(\mathbf{x})$  // see Equation 4.1
4:    $r \leftarrow h(\mathbf{x}')$ 
5:    $k_\alpha \leftarrow \text{concatenate}(p_x, r, y)$ 
6:    $k_\beta \leftarrow g(\mathbf{x}')$ 
7:    $k_\gamma \leftarrow \text{concatenate}(p_y, r)$ 
8:   if  $PDB_{L_m}[k_\alpha]$  is None then
9:      $PDB_{L_m}[k_\alpha] \leftarrow$  new Hashtable  $H$ 
10:  if  $PDB_{L_m}[k_\gamma]$  is None then
11:     $PDB_{L_m}[k_\gamma] \leftarrow$  new Set  $S$ 
12:  if  $PDB_{L_m}[k_\alpha][k_\beta]$  is None then
13:     $PDB_{L_m}[k_\alpha][k_\beta] \leftarrow (\mathbf{x}', y)$ 
14:    insert  $y$  into Set at  $PDB_{L_m}[k_\gamma]$ 
15:    insert  $r$  into  $R$ 
16: return  $R$ 
```

---

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  $(\mathbf{x}_n, y_n) \in D_m$  is ingested and buffered until a batch  $B$  has been accumulated. Then, the batch is preprocessed and inserted into the  $PDB_{L_m}$  as described in Algorithm 4. In words, the datapoint  $\mathbf{x}$  is scaled to the range  $[-1, 1]$  by applying a feature-wise min-max normalization, given by

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{X}_B)}{\max(\mathbf{X}_B) - \min(\mathbf{X}_B)} \cdot (b - a) + a, \quad (4.1)$$

where  $a = -1$ ,  $b = 1$  and  $\mathbf{X}_B$  is the set of data points within the batch  $B$ . After that, the scaled data point  $\mathbf{x}'$  is subject to both a locality-sensitive hashing function  $h$  and a non-cryptographic hashing function  $g$ . In this particular architecture,  $h$  is a gaussian random projection with a global seed for the initialization of the projection plane  $\mathbf{M}$  (see Section 2.2.3), such that regions  $r = h(\mathbf{x}')$  across local infrastructures are comparable.

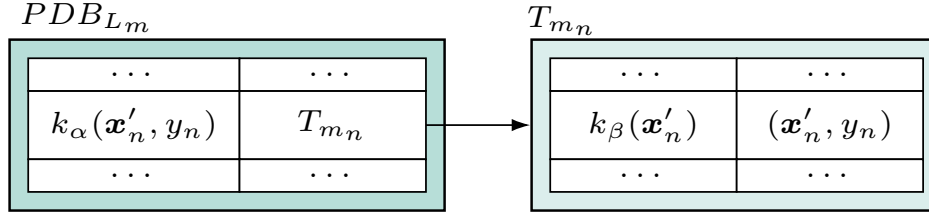


Figure 4.4: Nested indexing in a local pattern database.

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. In fact, the pairs within a region are further partitioned into disjoint subsets according to the 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  $(\mathbf{x}', y)$ , two keys  $k_\alpha, k_\beta \in K$  are constructed (see Lines 5-6 in Algorithm 4). The key  $k_\alpha$  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  $k_\beta$  is the result of the non-cryptographic hashing function  $g(\mathbf{x}')$ , which serves as a mechanism for deduplicating identical  $\mathbf{x}'$ .

Additionally, a third key  $k_\gamma$  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,  $k_\gamma$  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 the slot  $PDB_{L_m}[k_\alpha]$ . Likewise, if the slot  $PDB_{L_m}[k_\gamma]$  is empty, a new set<sup>1</sup> is initialized and inserted. After that, it is checked if the slot  $H[k_\beta]$ , which in turn is placed in  $PDB_{L_m}[k_\alpha]$ , is empty. In the positive case, the pair  $(\mathbf{x}', y)$  is inserted into that slot and the region  $r$  is inserted into the set  $R$ . Otherwise, no data is inserted into the local pattern database and no region is added to  $R$ . After processing a batch, the set of updated regions  $R$  is emitted as events into  $C_{L_m}$ .

Since data duplicates are filtered before the insert operation in this algorithm, events are also not emitted unnecessarily multiple times if, for example, the same dataset is sent to the service repeatedly. In other words, the inserts are idempotent, which is an important property in this architecture. Since there are operations in subsequent services that are relatively computationally intensive, emitting events is expensive. For this reason, such a streaming application might in practice implement buffers at regular intervals to collect and aggregate the events of several successive batches.

<sup>1</sup>A set describes a data structure which is essentially an unordered collection with no duplicate elements.

## 4.4 Complexity Estimation

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, the existence of multiple classes indicates a more complex 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  $\mathbf{M}$  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.  $\mathbf{x}_i \in D_1$  and  $\mathbf{x}_j^* \in D_2$  may be hashed to the same region  $h(\mathbf{x}_i) = h(\mathbf{x}_j^*)$ . However, it is also possible that the corresponding labels  $y_i \in D_1$  and  $y_j^* \in D_2$  are not equal and therefore lead to a different global view on that region's complexity state. Given that, the complexity estimation module acts as a bridge between the local and global components and answers the question, which regions are considered to be complex in a global context.

---

**Algorithm 5** Creating a global complexity state by combining local complexity states

---

**Input:** Regions  $R_{\text{in}}$

**Output:** Regions  $R_{\text{out}}$

```

1:  $R_{\text{out}} \leftarrow \text{new Set}$ 
2:  $m \leftarrow \text{getID}()$  // current local infrastructure identifier
3: for each  $r$  in  $R$  do
4:    $k_\gamma \leftarrow \text{concatenate}(p_y, r)$ 
5:    $Y_r \leftarrow PDB_{L_m}[k_\gamma]$ 
6:    $k_\delta \leftarrow \text{concatenate}(p_y, r, m)$ 
7:    $PDB_G[k_\delta] \leftarrow Y_r$ 
8:    $S \leftarrow \text{new Set}$ 
9:   for each  $m$  in  $\{1, \dots, M\}$  do
10:     $k_\delta \leftarrow \text{concatenate}(p_y, r, m)$ 
11:     $Y_r \leftarrow PDB_G[k_\delta]$ 
12:    insert  $Y_r$  into  $S$ 
13:   if  $|S| > 1$  then
14:      $c_r \leftarrow 1$ 
15:   else
16:      $c_r \leftarrow 0$ 
17:    $k_\kappa \leftarrow \text{concatenate}(p_c, r)$ 
18:   if  $PDB_G[k_\kappa] \neq c_r$  then
19:      $PDB_G[k_\kappa] \leftarrow c_r$ 
20:   insert  $r$  into  $R_{\text{out}}$ 
21: return  $R_{\text{out}}$ 

```

---

First, a set of regions  $R$  is received. Then, for each region  $r \in R$  the following operations are defined. The set of unique labels  $Y_r$  for a particular region has been stored in Algorithm 4 as auxiliary metadata, which is now retrieved by constructing the corresponding key  $k_\gamma$ . Subsequently, the slot  $PDB_{L_m}[k_\gamma]$  is accessed and  $Y_r$  is retrieved. In the next step,  $Y_r$  has to be stored in the  $PDB_G$ . Therefore, another key  $k_\delta$  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.

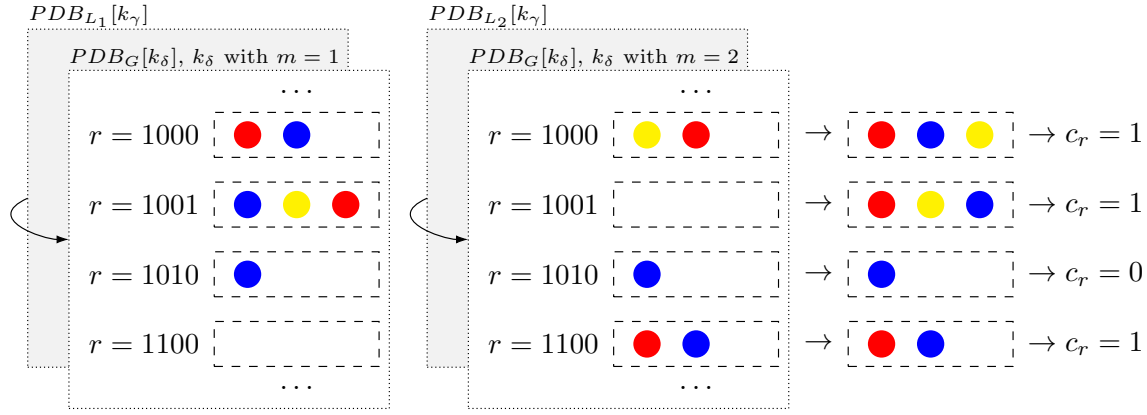


Figure 4.5: Illustration of the complexity estimation algorithm with two local infrastructures. Each local label set at  $PDB_{L_m}[k_\kappa]$  is synchronized into the global pattern database at  $PDB_G[k_\delta]$ , where  $k_\delta$  is constructed with  $m$ . Per region, the union operation is applied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

After  $Y_r$  is stored on a global level at  $PDB_G[k_\delta]$ , 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  $k_\delta$  are constructed. Each key retrieves the specific label set  $Y_r$  of a member and inserts its content into the set  $S$ .

After collecting all label sets in  $S$ , the complexity state is obtained by simply evaluating the cardinality  $|S|$ . 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  $k_\kappa$  is created by concatenating the prefix constant  $p_c$  and the region  $r$ . Note, that  $PDB_G[k_\kappa]$  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  $R_{out}$ , which is subsequently sent into the global event channel  $C_G$  in order to inform services in all local infrastructures about the update.



## 4.5 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 exchange medium for information. More specifically, multiple *Gaussian Mixture Models* (GMMs) are fitted on each label-subset of a region's data, which was stored in the indexing step in Section 4.3. 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. The purpose of that elaborate process is that synthetic data can be sampled from these models. 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.

---

### Algorithm 6 Retrieve Dataset from Region (Main Procedure)

---

**Input:** Regions  $R_{in}$

**Output:** Regions  $R_{out}$

```

1:  $m \leftarrow \text{getID}()$ 
2: for each  $r$  in  $R_{in}$  do
3:    $k_\kappa \leftarrow \text{concatenate}(p_c, r)$ 
4:    $k_\delta \leftarrow \text{concatenate}(p_y, r, m)$ 
5:    $c_r \leftarrow PDB_G[k_\kappa]$ 
6:    $Y_r \leftarrow PDB_G[k_\delta]$ 
7:   if  $c_r = 0$  then
8:     for each  $y$  in  $Y_r$  do
9:        $k_\omega \leftarrow \text{concatenate}(p_d, r, y, m)$ 
10:      delete model in  $PDB_G[k_\omega]$ 
11:   else
12:      $L \leftarrow \text{new List}$ 
13:     for each  $y$  in  $Y_r$  do
14:        $k_\alpha \leftarrow \text{concatenate}(p_x, r, y)$ 
15:        $H \leftarrow PDB_{L_m}[k_\alpha]$ 
16:       append  $H$  to  $L$ 
17:      $D \leftarrow \text{preprocessing}(L)$ 
18:     for each  $y$  in  $Y_r$  do
19:        $\text{GMM} \leftarrow \text{modelSelection}(D, y)$ 
20:        $k_\omega \leftarrow \text{concatenate}(p_d, r, y, m)$ 
21:        $PDB_G[k_\omega] \leftarrow \text{GMM}$ 

```

---

First, regions that have been updated are received as events. As the data is further organized per label within a region, the labels for a region are retrieved. 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 has been changed from complex to simple.

But if the region is complex, the generative model fitting procedure is triggered. Even if there are already models for the corresponding combination of region and label, an update of these models is initialized. For that, every hash table within a region, each containing data with a common label, is collected and added to a list. Subsequently, preprocessing operations prepare the collected region data for the model fitting. Details on the data preparation are described in Algorithm 7.

After that, the Model Selection process is started sequentially for each available label in the region. That way, the models are only fitted on data with the label in focus but evaluated with the complete region data. Specifics on the model selection are elaborated in Algorithm 9. Finally, the best fitted model is stored in the global pattern database. So far, the main procedure has been outlined. Next, the details on the data preprocessing and the model selection are elaborated.

---

**Algorithm 7** Preprocess Data
 

---

**Input:** List of HashMaps  $L$

**Output:** Dataset  $D$

```

1:  $D \leftarrow \text{getValues}(L)$ 
2: for each label in  $D$  do
3:   if label  $\neq 0$  then
4:     // split into binary
5:      $(X_a, y_a) \leftarrow (X, y)$  where  $y = \text{label}$ 
6:      $(X_b, y_b) \leftarrow (X, y)$  where  $y \neq \text{label}$ 
7:     if  $X.\text{shape}[0] < X.\text{shape}[1]$  then
8:        $X_a, y_a \leftarrow \text{upsample}(X_a, y_a)$ 
9: return  $(X_a, y_a), (X_b, y_b)$ 

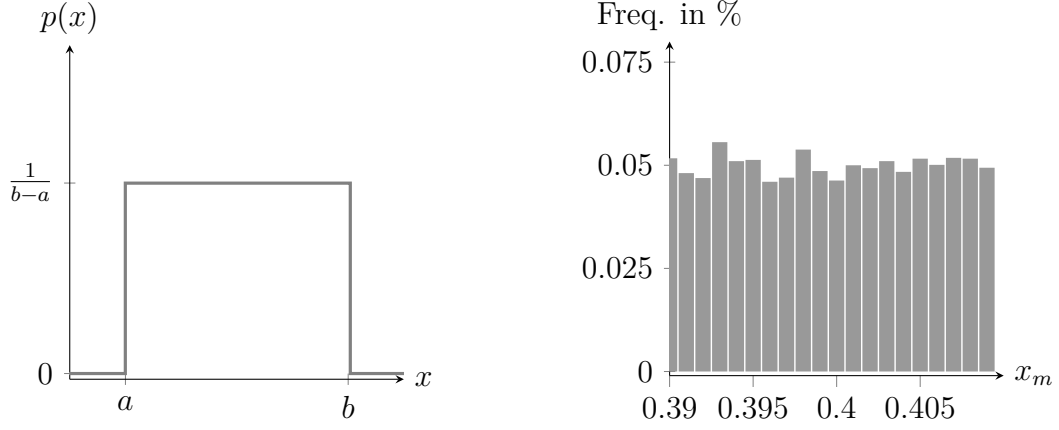
```

---

Starting with the preprocessing.

Splitting the dataset into a binary problem, such that the label that is currently in focus is the normal class and all other classes are attack classes. That way, the generative model can be evaluated using the machine learning efficacy method in the later course.

The upsampling process ensures that the fitting algorithm for the GMM is able to work. In practice, the number of samples has to be at least equal to the number of components. As some label subsets of a region may exhibit a low number of samples, in extreme cases only a single sample, an upsampling process is implemented. In particular, a set of nearest neighbours is generated per sample. First, the number of samples to generate is determined by the difference of the dimensionality of the data and the number of data points. Then, in order to generate from each data point equivalently, the number of nearest neighbours to sample is determined per data points. That is, the complete number of points to resample divided by the number of samples with an interger division (resulting in an integer). In case, the result of the interger division is zero, one is added to the result. Then for each data point  $x$  in the set  $X$ , nearest neighbours are generated by sampling from a uniform distribution. This is done by considering each feature value of  $x$  individually, such that the nearest neighbour is the concatenation of the sampled feature values as  $x^* = [p(x_1), p(x_2), \dots, p(x_M)]$ , where  $p(x_m) = \frac{1}{b-a}$  within the interval  $[x_m - \delta, x_m + \delta]$ . The value  $\delta$  controls the interval of the uniform distribution. The larger the value for  $\delta$ , the further the newly generated values deviate from the original feature values. In Figure 4.6b the value for a single feature  $x_m$



(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  $[x - \delta, x + \delta]$  where  $x = 0.4$  and  $\delta = 1 \cdot 10^{-2}$ .

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

is drawn uniformly at random. The original value of the feature was  $x_m = 4$ , which was extended by  $\delta = 1 \cdot 10^{-2}$ . By choosing a relatively small value for  $\delta$ , it is ensured that the generated nearest neighbours do not alter the original data distribution significantly, while enabling the subsequent fitting of the GMM for that set of data points.

---

#### Algorithm 8 Upsampling

---

**Input:** Collection of datapoints  $X$

**Output:** Upsampled collection of datapoints  $X_{up}$

```

1: nResample  $\leftarrow X.shape[1] - X.shape[0]$  // Difference of  $\dim(X)$  and  $\text{num}(X)$  to fill

2: nResamplePerX  $\leftarrow nResample // X.shape[0] + 1$ 
3:  $X^* \leftarrow \emptyset$ 
4: for each  $x$  in  $X$  do
5:    $x^* \leftarrow \text{new Array}$ 
6:   for each  $x_m$  in  $x$  do
7:     sample a new  $x_m^*$  from  $p(x_m)$  and insert into  $x^*$ 
8:   add  $x^*$  to  $X^*$ 
9:  $X_{up} \leftarrow X \cup X^*$ 
10: return  $X_{up}$ 

```

---

In the next phase of the algorithm, one or more GMMS are selected for a region's data, depending on the number of unique labels within. For each label subset, multiple models are fitted within a selection process. Since the resource demands of the EM algorithm for fitting a GMM are relatively complex, the data's dimensionality is reduced by applying a principal component analysis. Later in the course, when sampling data, the inverse operation using the same PCA parameters is applied on the synthetic data and bring it back into the original dimensions. Therefore, for each label subset of a region, both the parameters of a GMM and a PCA model is stored in the global pattern database.

Apply PCA on  $X_a$ , such that 99.9% of the variance of the data is preserved, then fit multiple GMM with the same data but with different parameters by the following rules; collect a list of different parameters for the number of components  $C = \{2k + 1 : k \in \mathbb{N}, 1 \leq k \leq K\}$  with  $K = \lfloor M/2 \rfloor$ , the different number of components is heuristically determined; in general, there is no exact method to determine the optimal number of components for a given dataset before fitting the model; thus, different parameters have to be tried out in a model selection method; Moreover, as stated in Section (section of GMM), the runtime of a single step of the EM algorithm in this setting is asymptotically  $O(NKd^3)$  or  $O(NKd^2)$  by using the incremental algorithm proposed in [PE15] ( $N$  data points,  $K$  components and  $d$  dimensions); therefore, this curse of dimensionality that is encountered in calculating the covariance matrix while fitting the GMM is coped by approximation; using the diagonal covariance matrix as approximation to the regular covariance matrix; moreover, the full covariance matrix can result in overfitting, especially on small datasets, whereas the diagonal approximation acts as a type of regularization to the model. Since the focus of this algorithm is mainly focused on providing the best model, the runtime is treated as a secondary factor; thus, both types of covariances are used within the model selection process  $\text{cov} = \{\text{"full"}, \text{"diagonal"}\}$ .

After fitting a model on  $X'_a = \text{PCA}(X_a)$ , the first metric for the selection is calculated. Precisely, the bayesian information criterion (BIC) is calculated as in Equation(X). Note, that for the case of a diagonal covariance matrix, the number of parameters to estimate change from  $\frac{d(d+1)}{2}$  to  $d$ , such that the BIC is given as

$$\text{BIC}(M|D) = (Kd + d + K - 1) \ln(N) - 2 \ln(\hat{L}). \quad (4.2)$$

After that, another evaluation metric is calculated. This process is called adversary evaluation and is based on the principles of generative adversarial networks, where a discriminator function, i.e., classifier, tries to distinguish between the original and synthetic data. In other words, the generative algorithm synthesizes fake data, which resembles the original data. Then, a dataset is assembled by combining fake data with class labels 0 and original data with class labels 1. 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 with a decision boundary (dashed line). In the second scenario (a), the generative model does a good job by creating synthetic data that could not be separated from the original data.

In order to quantify the performance of the discriminator, a classification metric is calculated. Since the original and synthetic data is equal in size and the posed problem is binary, the accuracy score is a precise metric, calculated as

$$a = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.3)$$

For each iteration in the model selection process, both the BIC and the accuracy score

**Algorithm 9** Model Selection

---

**Input:** Dataset  $D$  with  $X_a, y_a, X_b, y_b$   
**Output:** Tuple (GMM, PCA) containing model parameters

- 1:  $X'_a \leftarrow \text{PCA}(X_a)$
- 2:  $C \leftarrow \{2k + 1 : k \in \mathbb{N}, 1 \leq k \leq K\}$  with  $C = \lfloor M/2 \rfloor$
- 3:  $B \leftarrow$  new Array
- 4:  $\text{COV} \leftarrow \{\text{"full"}, \text{"diagonal"}\}$
- 5: **for each**  $k$  in  $C$  **do**
- 6:     **for each** cov in  $\text{COV}$  **do**
- 7:          $\text{GMM} \leftarrow \text{fitGMM}(X'_a)$
- 8:          $\text{GMMs}[kC] \leftarrow \text{GMM}$
- 9:          $b_{kC} \leftarrow \text{BIC}(\text{GMM}|X'_a)$
- 10:         $\text{acc} \leftarrow \text{AdversaryEvaluation}(X, y, \text{GMM}, \text{PCA})$
- 11:         $B[kC] \leftarrow (b_{kC}, \text{acc})$
- 12: sort  $B$  by  $b_{kC}$  in  $B[0]$
- 13: sort  $B$  by  $\text{acc}$  in  $B[1]$
- 14:  $b_{kC} \leftarrow B[-1]$
- 15: **return** ( $\text{GMMs}[kC], \text{PCA}[kC]$ )

---

is stored. After all models have been trained on all parameter combinations, a combined metric is calculated from the stored metrics. In order to combine BIC and accuracy, the BIC scores have to be normalized into the range  $[0, 1]$  by applying the min-max-normalization from Equation 4.1 to the set of BIC scores. After that, the mean is calculated for each metric pair as

$$m = \frac{\text{BIC} + a}{2}. \quad (4.4)$$

Finally, the model with the lowest combined metric  $m$  is selected. This way, both metrics are equally utilized leading to a balanced theoretical (BIC) and practical (accuracy) guarantee of model fitness. Both the GMM and the PCA parameters are stored in the global pattern database. In numbers, these can range, depending on which type of covariance matrix is chosen for the GMM as already shown in Equation X for the full covariance matrix and in Equation X for the diagonal case. 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, leading to a number of  $D \times M$  elements to store. Therefore, per region, the number of parameters to store is at least

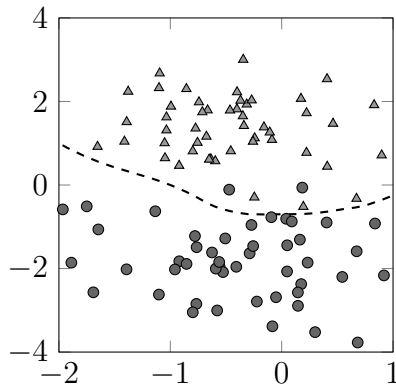
$$n_{y_r} * (Kd + d + K - 1 + Md) \quad (4.5)$$

and at most

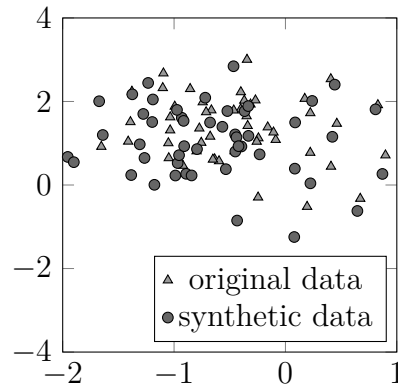
$$n_{y_r} * ((Kd + K(d(d + 1)/2) + K - 1) + Md). \quad (4.6)$$

**Algorithm 10** Adversary Evaluation**Input:**  $X_r, y_r$ , GMM, PCA**Output:** acc from DecisionTree Model  $DT$ 

- 1:  $X'_s \leftarrow$  sample  $|X_r|$  data points from  $GMM$
- 2:  $X_s \leftarrow \text{PCA}(X'_s)^{-1}$
- 3:  $y_s \leftarrow \{1\}$  // as many 0s as the number of  $X_b$
- 4:  $y_r \leftarrow 0$  // setting all real labels to 0
- 5:  $X, y \leftarrow \text{concatenate}(X_s, X_r, y_s, y_r)$
- 6:  $X_{\text{train}}, y_{\text{train}}, X_{\text{test}}, y_{\text{test}} \leftarrow \text{shuffleSplit}(X, y)$
- 7:  $DT \leftarrow \text{fitDecisionTree}(X_{\text{train}}, y_{\text{train}})$
- 8:  $\tilde{y} = \{DT(x_1), \dots, DT(x_n)\} \leftarrow$  predict  $X_{\text{test}}$  with  $DT$
- 9:  $F1 \leftarrow \text{f1}(y_{\text{test}}, \tilde{y})$
- 10: **return** F1



(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: separation of original and synthetic data.

## 4.6 Classifier Fitting

The objective of the classifier fitting service is to provide the local IDS with the latest classification model trained on both original local data and synthetic global data. After each change to the generative models in the global pattern database made by the previous service, an event is sent into the communication channel. Within a defined time window, all such events are collected. If at least a single event was collected, the classifier fitting service is initiated and begins with assembling a training dataset. For this task, all original data are first loaded from the local PDB. Then, all generative models are loaded sequentially from the PDB and provided. In the process, synthetic data is generated in the amount that was also originally found in the corresponding region. After each sampling, the synthetic data is transferred from the lower-dimensional subspace to the original data space by the inverse function of the original PCA operation. Subsequently, the normalization operation is also reversed for both the original and the synthetic data by loading the corresponding parameters that were used to normalize a batch from the PDB.

---

**Algorithm 11** Dataset Assembly and Classifier Fitting

---

**Input:** Region Update Event  $r$

**Output:** Model Update Notification

---





# Chapter 5

## Experimental Evaluation

### 5.1 Flow Feature Analysis

#### 5.1.1 Datasets

#### 5.1.2 Feature Extraction and Preprocessing

### 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



# Chapter 6

## Conclusion and Outlook

### 6.0.1 Conclusion

### 6.0.2 Outlook



# Bibliography

- [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.

- [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 thirty-fourth annual ACM symposium on Theory of computing*. 2002, pp. 380–388.
- [CM02] Frédéric Cuppens and Alexandre Mieke. “Alert correlation in a cooperative intrusion detection framework”. In: *Proceedings 2002 IEEE symposium on security and privacy*. IEEE. 2002, pp. 202–215.
- [GKM03] Ayalvadi J Ganesh, A-M Kermarrec, and Laurent Massoulié. “Peer-to-Peer Membership Management for Gossip-Based Protocols”. In: *IEEE Transactions on Computers* (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).
- [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.
- [Dat+04] Mayur Datar et al. “Locality-sensitive hashing scheme based on p-stable distributions”. In: *Proceedings of the twentieth annual symposium on Computational geometry - SCG '04*. Brooklyn, New York, USA: ACM Press, 2004, p. 253. ISBN: 978-1-58113-885-6. DOI: [10.1145/997817.997857](https://doi.org/10.1145/997817.997857). URL: <http://portal.acm.org/citation.cfm?doid=997817.997857> (visited on 01/14/2022).
- [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>.
- [Loc+05] M. E. Locasto et al. “Towards Collaborative Security and P2P Intrusion Detection”. In: *IEEE SMC Information Assurance Workshop* (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.
- [Das+06] Denver Dash et al. “When Gossip is Good: Distributed Probabilistic Inference for Detection of Slow Network Intrusions”. In: *AAAI National Conference on Artificial Intelligence* (2006).

- [Dum+06] Claudiu Duma et al. “A Trust-Aware, P2P-Based Overlay for Intrusion Detection”. In: *International Workshop on Database and Expert Systems Applications* (2006).
- [Kor06] Jesse Kornblum. “Identifying almost identical files using context triggered piecewise hashing”. en. In: *Digital Investigation* 3 (Sept. 2006), pp. 91–97. ISSN: 17422876. DOI: [10.1016/j.diin.2006.06.015](https://doi.org/10.1016/j.diin.2006.06.015). URL: <https://linkinghub.elsevier.com/retrieve/pii/S1742287606000764> (visited on 01/21/2022).
- [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.
- [VF06] Vivek Vishnumurthy and Paul Francis. “On Heterogeneous Overlay Construction and Random Node Selection in Unstructured P2P Networks”. In: *IEEE International Conference on Computer Communications* (2006).
- [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.
- [BA08] Rainer Bye and Sahin Albayrak. *CIMD - Collaborative Intrusion and Malware Detection*. 2008.
- [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.
- [Rou10] Vassil Roussev. “Data Fingerprinting with Similarity Digests”. en. In: *Advances in Digital Forensics VI*. Ed. by Kam-Pui Chow and Sujeet Sheno. Vol. 337. Series Title: IFIP Advances in Information and Communication Technology. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 207–226. DOI: [10.1007/978-3-642-15506-2\\_15](https://doi.org/10.1007/978-3-642-15506-2_15). URL: [http://link.springer.com/10.1007/978-3-642-15506-2\\_15](http://link.springer.com/10.1007/978-3-642-15506-2_15) (visited on 01/21/2022).
- [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.
- [Gil+13] Manuel Gil Pérez et al. “RepCIDN: A Reputation-Based Collaborative Intrusion Detection Network to Lessen the Impact of Malicious Alarms”. In: *Journal of Network and Systems Management* (2013).
- [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/CB09781139924801.004](https://doi.org/10.1017/CB09781139924801.004).
- [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.
- [PE15] Rafael Coimbra Pinto and Paulo Martins Engel. “A fast incremental gaussian mixture model”. In: *PloS one* 10.10 (2015), e0139931.
- [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/>.
- [MH17] Vitor Hugo Galhardo Moia and Marco Aurélio Amaral Henriques. “Similarity Digest Search: A Survey and Comparative Analysis of Strategies to Perform Known File Filtering Using Approximate Matching”. en. In: *Security and Communication Networks* 2017 (2017), pp. 1–17. ISSN: 1939-0114, 1939-0122. DOI: [10.1155/2017/1306802](https://doi.org/10.1155/2017/1306802). URL: <https://www.hindawi.com/journals/scn/2017/1306802/> (visited on 01/24/2022).
- [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.
- [WM18] Michael E. Whitman and Herbert J. Mattord. *Principles of information security*. Cengage Learning, 2018. ISBN: 978-1-337-10206-3.
- [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=nT4QEAAQBAJ>.
- [Cor+22] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.
- [Pha] Peter Phaal. *sFlow Specification Version 5*. [https://sflow.org/sflow\\_version\\_5.txt](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 Tables

4.1	Summary of the architecture notation. . . . .	40
-----	---	----



# List of Figures

2.1	A categorization of intrusions into different subtypes . . . . .	5
2.2	A taxonomy for IDS that distincts systems based on the utilized detection method or the scope in which it operates . . . . .	5
2.3	test . . . . .	7
2.4	In the $cR$ -approximate nearest neighbour problem some point within $cR$ is accepted, if there exists a point $p$ where $d(p, q) \leq R$ . . . . .	13
2.5	The behaviour of conventional hashing and LSH . . . . .	14
2.6	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. . . . .	15
2.7	Illustration of a random hyperplane (red) partitioning the space . . . .	18
2.8	Visual Proof of claim in equation 2.14 . . . . .	19
2.9	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 . . . . .	21
2.10	Marginals $p(x_M)$ , $p(x_N)$ and Conditional Gaussian $p(x_{M N})$ . . . . .	22
4.1	Integration of the approach into a generic NIDS. . . . .	36
4.2	Building the global view by combining $M$ local views. . . . .	38
4.3	High level CIDS architecture. . . . .	39
4.4	Nested indexing in a local pattern database. . . . .	42
4.5	Illustration of the complexity estimation algorithm with two local infrastructures. Each local label set at $PDB_{L_m}[k_\kappa]$ is synchronized into the global pattern database at $PDB_G[k_\delta]$ , where $k_\delta$ is constructed with $m$ . Per region, the union operation is applied on the global label sets, whereupon the complexity is derived from. . . . .	44
4.6	Nearest neighbours are generated by sampling each feature value from a the uniform distribution. . . . .	47
4.7	Adversary evaluation: separation of original and synthetic data. . . . .	50



# List of Algorithms

1	LSH Preprocessing . . . . .	16
2	LSH Query . . . . .	17
3	GMM Selection with BIC . . . . .	29
4	Preprocessing and inserting $B \subset D_m$ into $PDB_{L_m}$ . . . . .	41
5	Creating a global complexity state by combining local complexity states	43
6	Retrieve Dataset from Region (Main Procedure) . . . . .	45
7	Preprocess Data . . . . .	46
8	Upsampling . . . . .	47
9	Model Selection . . . . .	49
10	Adversary Evaluation . . . . .	50
11	Dataset Assembly and Classifier Fitting . . . . .	51