Anomaly Detection
in
Cloud Computing Environments

vorgelegt von
M. Sc.
Florian Johannes Schmidt

an der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. David Bermbach
Gutachter: Prof. Dr. Odej Kao
Gutachter: Prof. Dr. Johan Tordsson
Gutachter: Prof. Dr. Jan Nordholz

Tag der wissenschaftlichen Aussprache: 18. Juni 2020

Berlin 2020

Thank you!

{ Kathrin, Martina, Hartmut, Friederike, Maryse, Einstein, Felix, Fabian, Fiona, Marion, Thomas, Susanne, Heino, Hildegard, Ella, Harry, Helmut, Anton, Marcel, Odej, Alex, Sören, Ilya, Lauritz, Sasho, Jana, Feng, Tobias, Jan, Kevin, Stefan, Steven, Annika, Alexandra, Elisa, Eva, Paul, René, Sven, Vincent, Johannes, Yannick, Tim }

Zusammenfassung

Cloud Computing Paradigmen, werden in der modernen Softwareentwicklung bereits von den meisten Unternehmen angewendet. Die Bereitstellung von digitalen Diensten in einer Cloudumgebung bietet sowohl die Möglichkeit der kosteneffizienten Nutzung von Ressourcen als auch die Möglichkeit auf Bedarf dynamisch die Anwendungen zu skalieren. Basierend auf dieser Flexibilität werden immer komplexere Softwareanwendungen entwickelt, welches zu anspruchsvollen Wartungsarbeiten der Gesamtinfrastruktur führen. Ebenfalls werden immer höhere Ansprüche an die Verfügbarkeit von Softwarediensten gestellt (99,999% im Industriekontext), was durch die Komplexität moderner Systeme nur noch schwieriger und unter großer Mühe gewährleistet werden kann. Aufgrund dieser Trends steigt der Bedarf an intelligenten Anwendungen, die automatisiert Anomalien erkennen und Vorschläge erarbeiten, um Probleme zu erkennen, zu beheben oder zumindest zu mindern um keinen negativen Einfluss auf die Servicequalität zu kaskadieren.

Diese Arbeit beschäftigt sich mit der Erkennung von degradierten abnormalen Systemzuständen in Cloudumgebungen. Hierbei wird sowohl eine holistische Analysepipeline und -infrastruktur beschrieben als auch die Anwendbarkeit von verschiedenen Strategien des maschinellen Lernens diskutiert, um möglichst eine voll automatisierte Lösung bereitzustellen. Basierend auf den zugrunde liegenden Annahmen, wird ein neuartiger unsupervised Anomalieerkennungsalgorithmus namens CABIRCH vorgestellt und dessen Anwendbarkeit analysiert und diskutiert. Da die Wahl der Hyperparameter einen wichtigen Einfluss auf die Genauigkeit des Algorithmus hat, wird zudem ein Hyperparameterauswahlverfahren mit einer neuartigen Fitness-Funktion vorgestellt, welches zur Vollautomatisierung der Anomalieerkennung führen soll. Hierbei ist das Verfahren generalisiert anwendbar für eine Vielzahl von unsupervised Anomalieerkennungsalgorithmen, welche basierend auf jüngsten Veröffentlichungen umfassend evaluiert werden. Dabei wird die Anwendbarkeit zur automatisierten Erkennung von degradierten abnormalen Systemzuständen gezeigt und mögliche Limitierungen diskutiert. Die Ergebnisse zeigen, dass eine Erkennung der verschiedenen Anomalien gewährleistet werden kann, jedoch mit einer Fehlalarmrate von über 1%.

Abstract

Cloud computing is widely applied by modern software development companies. Providing digital services in a cloud environment offers both the possibility of cost-efficient usage of computation resources and the ability to dynamically scale applications on demand. Based on this flexibility, more and more complex software applications are being developed leading to increasing maintenance efforts to ensure the reliability of the entire system infrastructure. Furthermore, highly available cloud service requirements (99.999% as industry standards) are difficult to guarantee due to the complexity of modern systems and can therefore just be ensured by great effort. Due to these trends, there is an increasing demand for intelligent applications that automatically detect anomalies and provide suggestions solving or at least mitigating problems in order not to cascade a negative impact on the service quality.

This thesis focuses on the detection of degraded abnormal system states in cloud environments. A holistic analysis pipeline and infrastructure is proposed, and the applicability of different machine learning strategies is discussed to provide an automated solution. Based on the underlying assumptions, a novel unsupervised anomaly detection algorithm called CABIRCH is presented and its applicability is analyzed and discussed. Since the choice of hyperparameters has a great influence on the accuracy of the algorithm, a hyperparameter selection procedure with a novel fitness function is proposed, leading to further automation of the integrated anomaly detection. The method is generalized and applicable for a variety of unsupervised anomaly detection algorithms, which will be evaluated including a comparison to recent publications. The results show the applicability for the automated detection of degraded abnormal system states and possible limitations are discussed. The results show that detection of system anomaly scenarios achieves accurate detection rates but comes with a false alarm rate of more than 1%.

Contents

Chapter 1

Introduction

Digitalization transforms our world in various areas like Industry 4.0 (product line, manufacturing automation, predictive maintenance, etc.), transportation (self-driving cars, car-2-car communication, intelligent traffic control systems), smart home, medical assisted surgery, and many more. Gartner predicts that by 2020 there exist more than 20 billion connected IoT-devices [1]. Cisco even forecasts 28.5 billion connected devices by 2022 [2]. While the number of devices and sensors increases, the key network technologies like 5G and virtualization of cloud computing and fog computing change the business opportunities and enable flexibility for the infrastructure. The improved flexibility and business opportunities come at a high cost, as the system complexity increases significantly. It introduces the challenge of not only administering the complex IT-infrastructure but also adds the challenges of maintaining every e.g. remote device, edge cloud, software service, and the heterogeneous networks in between. Managing this complexity surpasses the ability of human experts to oversee the entire system and react with quick responses to meet the promised Quality of Service (QoS) parameters or even Service Level Agreements (SLA).

In application scenarios such as softwarization of dedicated hardware solutions to virtualized environments, telecommunication providers hope to benefit from increased flexibility and cost-effectiveness. Still, the given dedicated hardware components provide a reliability of 99.999% [3], which is therefore demanded for the virtualized components. Due to the increased complexity of the computation model, which includes hardware components and a stack of virtualized components, softwarized components cannot cope with the high demand for reliability. Because of the fragility of such system stacks, the expectations of system administrators increases to maintain the continuous operation of the services [4]. With respect to this and the recent developments of artificial intelligence (AI), concepts are developed to analyze and automate large portions of operational tasks for administrators. AI-based automation of infrastructure operation (AIOps) provides the vision of establishing a system able to autonomously operate and remediate large IT environments.

The increased demand for qualified operation maintenance support reflects the establishment of creating new job positions like DevOps or Site Reliability Engineers (SRE). The four pillars of effective DevOps within any company assume communication management among teams (Collaboration and Affinity), but also the introduction and usage of helpful tools as well as scaling to the organization's needs [5]. For example, Google coined the term SREs with specific principles to strategically implement the four pillars of DevOps, assuming the mindset that the developed code can provide problems at any time [6]. Gartner also provides four phases of IT-operations [7] including descriptive IT, anomaly detection and diagnostics, proactive operations, and avoidance of high-severity outages. In order to provide DevOps and SREs helpful

guidance, machine learning can help to provide solutions to optimize such operational workflows.

Nowadays, active and passive monitoring tools for measuring the system's behavior and QoS performance are widely used by administrators. As a consequence, automatic alerts are integrated and triggered by expert-based fixed thresholds for these measurements in order to inform the administrators about failures and anomalies. This helps to identify faster problematic system states, which may cause a decrease in QoS. Even though these tools already support the administrators, it often remains time-consuming to find the correct root causes of problems as well as determining the correct counter measurements in order to avoid a drop of the QoS.

The dynamics of systems additionally increase complexity. Nowadays, systems are quickly scaled up and down due to load changes by e.g. increase of user activity through the day, as well as the agile development of software introducing changes in e.g. microservice behavior. Especially for maintaining service specific thresholds, which meet the border between abnormal and normal behavior, it gets increasingly more difficult. On the one hand, it depends on the individual service or operating system's resource usages and on the other hand on the environmental influences (system load caused or introduced by users or surrounding virtualized infrastructure). Furthermore, not only the determination of a suitable threshold is difficult, but as the system behavior changes over time, the change of such thresholds has to be performed continuously in order to provide anomaly alerts as soon as possible to provide the chance to react before a component fails. Thus, automated mechanisms have to be developed to enable such support and, in the best case, further give valuable information to the administrator in real-time.

Finding an anomaly, before it causes a component to fail, is a challenging task as it is highly dependent on the system's individual behavior. Human administrators therefore need a lot of time to investigate large amounts of historic data to gain detailed insights to find the actual root cause. Then, they are able to start effective remediation counter actions, keeping the system highly reliable. Therefore, we proposed a holistic AIOps platform together with a self-healing analytics pipeline to automatically detect and remediate anomalous system components, helping current administrators to detect problems more efficiently and give recommendations to select accurate counter actions [8].

1.1 Research Objectives and Main Contributions

Main research objective: Investigate, develop and apply anomaly detection techniques to a holistic AIOps platform enhancing zero-touch administration for highly dependable IT-infrastructures. The objective concentrates on resource monitoring of black-box services, which can run on different levels within the IT environment.

The term black-box already indicates that the solution should be aware of any cloud service being monitored. Thus, the algorithmic component must cope with unknown services.

- **RO-1**: Formulate requirements for anomaly detection approaches to be applicable in productive environments and easily integrable into existing IT-infrastructure.
- **RO-2**: Investigate the applicability of supervised, semi-supervised, unsupervised techniques for productive usage within IT-infrastructures.
- **RO-3**: Develop, implement, and evaluate strategies for anomaly detection to meet the requirements towards a zero-touch administration.

This thesis concentrates on the definition and implementation of the anomaly detection component within a self-healing analytics pipeline, which is necessary for automated anomaly detection for data streams. As the anomaly detection component

is based on the holistic AIOps framework [8], we focus on the evaluation of the proposed methodologies on the domain of black-box service resource monitoring deployed on a replicated virtualized cloud infrastructure.

The key contributions of this thesis are focused on the following points:

- We describe the overall framework ZerOps, which is capable of decentralized execution of AI approaches in productive system cloud environments. Requirements and assumptions are defined to meet industry standards towards zero-touch administration. We analyze and discuss the applicability of different learning strategies for AI-based models and how they can be incorporated into the system design. Additionally, we present results of supervised anomaly detection as baseline results for the given domain.
- As supervised and semi-supervised detection approaches do not meet zero-touch requirements, we continue to develop an unsupervised detection capable of alerting e.g. unknown anomalies, adapting to concept changes, while learning and predicting in an online manner. We propose a novel concept adapting clustering algorithm, called CABIRCH, which can be applied to unsupervised anomaly detection and provides the required properties. The impact of hyperparameter settings as well as the applicability towards specific types of anomaly patterns are investigated.
- We define a generalized anomaly detection methodology, called IFTM. Based on the generalized IFTM methodology, we develop and evaluate a strategy to cope with the cold start problem. The cold start problem arises when newly monitored system components are deployed and further IFTM models must be initialized. In order to increase the quality of prediction results right from the beginning, a hyperparameter optimization is proposed utilizing a novel objective function for automated IFTM model configuration.
- Lastly, an extensive evaluation of several IFTM-based anomaly detection models is presented and discussed. The evaluation includes qualitative results from a production-ready cloud environment as well as quantitative runtime benchmarks in order to enable decision support when applying different methods in heterogeneous environments like IoT gateways or Edge-Cloud environments enabling decentralized computing.

1.2 Publications

Parts of this thesis and related contributions have been published in the following list of peer-reviewed articles:

[1] René Wetzig, Anton Gulenko, and Florian Schmidt. Unsupervised Anomaly Alerting for IoT-Gateway Monitoring using Adaptive Thresholds and Half-Space Trees. In *2019 IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 161–168. IEEE, 2019.

[2] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschläger, Alexander Acker, and Odej Kao. Unsupervised anomaly event detection for cloud monitoring using online arima. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 71–76. IEEE, 2018.

[3] Alexander Acker, Florian Schmidt, Anton Gulenko, and Odej Kao. Online density grid pattern analysis to classify anomalies in cloud and nfv systems. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 290–295. IEEE, 2018.

[4] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschläger, Alexander Acker, and Odej Kao. Unsupervised anomaly event detection for vnf service monitoring using multivariate online arima. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 278–283. IEEE, 2018.

[5] Marcel Wallschläger, Anton Gulenko, Florian Schmidt, Alexander Acker, and Odej Kao. Anomaly detection for black box services in edge clouds using packet size distribution. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–6. IEEE, 2018.

[6] Dora Szücs and Florian Schmidt. Decision tree visualization for high-dimensional numerical data. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 190–195. IEEE, 2018.

[7] Florian Schmidt, Anton Gulenko, Marcel Wallschläger, Alexander Acker, Vincent Hennig, Feng Liu, and Odej Kao. Iftm-unsupervised anomaly detection for virtualized network function services. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 187–194. IEEE, 2018.

[8] Anton Gulenko, Florian Schmidt, Alexander Acker, Marcel Wallschläger, Odej Kao, and Feng Liu. Detecting anomalous behavior of black-box services modeled with distance-based online clustering. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 912–915. IEEE, 2018.

[9] Florian Schmidt and Yannick Ehrenfeld. Vimec: Interactive application for micro-cluster visualizations. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Posters*, pages 29–31, Eurographics Association, 2018.

[10] Marcel Wallschläger, Anton Gulenko, Florian Schmidt, Odej Kao, and Feng Liu. Automated anomaly detection in virtualized services using deep packet inspection. *Procedia Computer Science*, 110:510–515, 2017.

[11] Alexander Acker, Florian Schmidt, Anton Gulenko, Reinhard Kietzmann, and Odej Kao. Patient-individual morphological anomaly detection in multi-lead electrocardiography data streams. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3841–3846. IEEE, 2017.

[12] Anton Gulenko, Marcel Wallschläger, Florian Schmidt, Odej Kao, and Feng Liu. A system architecture for real-time anomaly detection in large-scale nfv systems. *Procedia Computer Science*, 94:491–496, 2016.

[13] Anton Gulenko, Marcel Wallschläger, Florian Schmidt, Odej Kao, and Feng Liu. Evaluating machine learning algorithms for anomaly detection in clouds. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2716–2721. IEEE, 2016.

The full list of publications can be visited at `http://www.user.tu-berlin.de/flohannes/florianschmidt/`.

1.3 Outline of the Thesis

The rest of this thesis is structured as follows:

- Chapter 2 describes the background needed to understand key concepts of this thesis, including definitions about e.g. anomalies, key machine learning methodologies, and evaluation approaches.

- Chapter 3 presents related work of types of system anomaly patterns, anomaly detection methodologies, as well as an overview of online clustering methods and hyperparameter optimization.
- Afterwards, Chapter 4 provides further information about the holistic AIOps framework ZerOps, the inclusion of the anomaly detection within the self-healing pipeline. Furthermore, the chapter explains different supervised and semi-supervised anomaly detection approaches and discusses the applicability for productive usage. These results narrow down the expected requirements a holistic anomaly detection solution should be capable of.
- Based on these results, Chapter 5 presents the Concept Adapting BIRCH approach, showing necessary adaptations to the clustering technique BIRCH by Zhang et al. [9] in order to meet the proposed requirements for unsupervised anomaly detection.
- Chapter 6 provides solutions to cope with the cold start problem including a concept of autonomous hyperparameter optimization and applicability to several existing unsupervised anomaly detection models.
- Based on the existing methodology, Chapter 7 presents an evaluation for the evaluation setup with the black-box service use case and discusses qualitative and quantitative results as well as states future research, which can be conducted.
- Lastly, Chapter 8 summarizes and concludes the thesis.

Chapter 2

Background

2.1 Anomaly detection

There exist many different definitions for the term *anomaly detection* [10–17]. In this thesis, we refer to the definition by Chandola et al. [18]. Chandola et al. distinguish with their definition of normal and abnormal behavior through probability regions within a stochastic model.

Definition 2.1.1 (Anomaly detection [18])
Normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model.

For providing a clearer understanding of Definition 2.1.1 of anomalies with respect to data stream capabilities, we extend Chandola et al. definition by:

Definition 2.1.2 (Anomaly detection extension)
The stochastic model is context-dependent and can vary over time. Thus, anomalies are defined over the current context, which at least includes in the case of time series the context parameter time.

This extension is necessary due to the dynamics of a monitored system environment.

Through internal and external changes, the data stream behavior can change, which is referred to as concept drift. Jiang and Gruenwald [19] serve a suitable definition for this:

Definition 2.1.3 (Concept drift [19])
The data distribution in a data stream changes over time.

Anomaly detection algorithms should be robust to concept drifts if these model the normal behavior. Furthermore, it is challenging to the machine learning algorithm to distinguish accordingly between normal concept drifts and anomalies.

As Definition 2.1.1 suggests, for normally behaving data a stochastic model can be learned. Such a semi-supervised approach is related to an One-class classification task.

Definition 2.1.4 (One-class classification [20])
The task in One-class classification is to define a classification boundary around the positive (or target) class, such that it accepts as many objects as possible from the positive class, while it minimizes the chance of accepting non-positive (or outlier) objects.

As Khan and Madden [20] describe, the main challenge within this task is the definition of boundary around the learned class. The boundary should be compact to provide a proper border between normal and abnormal data points. This concept is used for anomaly detection as well as the definition of hyperparameter optimization for this work.

2.1.1 Types of Anomalies

Many publications suggest the three different types of anomalies: point, contextual and collective [21,22]. We follow the definition by Hodge and Austin [13] and Chandola et al. [18], who defined the three types as follows:

Definition 2.1.5 (Types of anomalies [13, 18])

1. *Isolated individual point in a dataset.*
2. *A data point that is isolated with respect to other data points in the context. Contextual attributes might be time, location, etc. Barnett and Lewis [11] define type 2 outliers as the additive outliers for time series data. The good thing about additive outliers is that they do not influence the other data points in context.*
3. *A particular group of data points with respect to the entire dataset. Barnett and Lewis [11] called them Innovations Outliers for time series data. The bad thing about innovations outliers is that they influence other data points of the same context and try to hide.*

Sadik and Gruenwald [23] state that for data streams, just type 2 or 3 anomalies can surface, but never type 1, because a time series has a temporal context with each data point. As the data streams are considered as a possibly infinite number of time-dependent data points. Their processing is therefore considered to be online as we cannot expect to store all data points. Therefore, at any particular moment, only a subset of the entire dataset is available like a group of an entire dataset. With respect to type 1, a data point cannot be described as an isolated individual point with respect to the entire dataset as the data points are connected through the context of time.

Ramchandran and Sangaiah [22] and Goldstein and Uchida [17] define furthermore *global* and *local* outliers. *Global* outliers are data points, which are abnormal compared to the complete (global) dataset. On the other hand, *local* outliers are data points within a local context of a subset of data.

While Sadik and Gruenwald [23] argue that there cannot exist the entire dataset at any point in time, they assume, that it is never possible to detect global anomalies. As just a subset of data from the latest temporal context are available, local anomalies can be detected, which we consider also as type 1 anomalies with respect to the local context.

2.1.2 Failures and Degraded State Anomalies

Within the IT-infrastructure context, there exist multiple meanings and wordings with respect to the term anomaly like faults, errors, failures, etc.

Avizienis et al. [24] define service faults to be the root cause of an error, while errors represent a deviation of the correct internal service component state. Errors may propagate to further internal service components through e.g. API calls and cause a service failure. Service failures are lastly defined by the presence of incorrect service state expectations to external service requests. The main focus of this thesis are errors on component-level, which are considered as anomalies in the following.

Failures can be caused by either hardware modules or software components. Hardware failures relate to specific hardware components of a computer system like hard

disks, memory modules, network cards, processors [25]. Gray [26], Barroso, and Hölzle [27] showed in their studies that hardware failures cause less than 10% of total service failures. Oppenheimer et al. [28] also indicated that the smallest number of contributing failures relate to hardware (10–25%).

In Gray's [26] study (more than 5 years of field data of highly fault-tolerant Tandem servers), 60% of fault events relate to software errors and 20% refer to maintenance and operations faults. Similar measurements are presented by Oppenheimer et al. [28], who studied service-level failures of three internet services consisting of more than 1500 servers. They concluded misconfiguration of service components, as well as operator-caused errors, are the largest portion contributing to service failures.

Barroso and Hölzle [27] presented the distribution of disruption events of Google's main services (6 weeks of field data) to be consistent with the above described studies. Service faults are mainly caused by 60% software and configuration errors and 20% due to human error and networking issues.

In addition, Barroso and Hölzle [27] introduced four categories for distinguishing service-level failures representing the severity (with respect to the QoS) in decreasing order:

- *Corrupted*: Stored state of the service is lost, corrupted and cannot be regenerated.
- *Unreachable*: The service is unreachable and does not respond to requests.
- *Degraded*: The service is available but operates in some degraded state.
- *Masked*: Faults occur in service components but are completely hidden to the entity using the service.

Masked failures are related to and caused by fault-tolerant operation and design of software and hardware components. With respect to the QoS, masked failures do not influence the QoS negatively, in contrast to the other categories.

Within this work, we consider degraded failures as well as masked failures as the aimed anomalies to be detected due to the following two reasons:

1. We assume that unreachable services and corrupted failures are easy to detect through e.g. active probing etc., while failures causing degraded state issues or even without any influence on the QoS are more difficult to detect.
2. In cases of degraded state and masked failures, we assume a larger impact of possibilities to remediate such problems before failures lead to unreachability or corruption.

Cotroneo et al. [29] described that system components operating in anomaly states suffer from performance degradation, whereby the degradation severity depends on the severity of the anomaly state. In order to provide self-healing capabilities, degraded state anomalies are the focus of this work as those anomalies allow the pipeline to react.

Additionally, we consider detecting not only anomalies on the larger service-system level but on a finer granular service-component level. This enables to alert administrators presenting at component-level anomalies.

In addition to failures, there exists the phenomenon of *software aging* [30], which has to be considered for anomaly detection. Over time, the failure rate increases for software due to the increasing chance that errors happen by e.g. unbounded resource consumption, data corruption, accumulation of numerical errors [31]. Avritzer and Weyuker [32] describe software aging for telecommunication switching software, which leads to gradual performance degradation. Software aging may cause aging-related failures [33], but also provides concept shift behavior when applying anomaly detection approaches as errors might be masked.

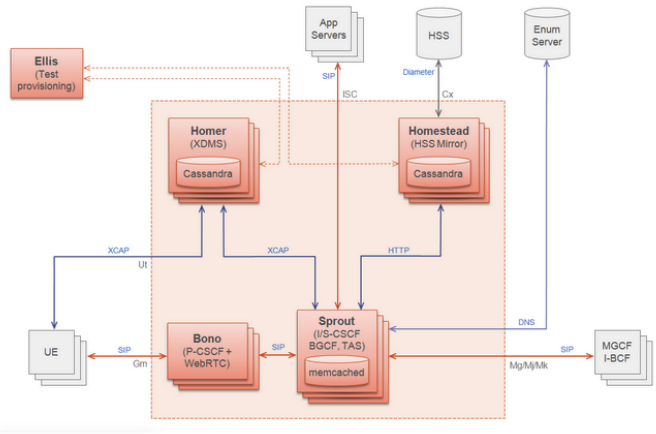


Figure 2.1: Clearwater service components and their connection between each other. Source: [36].

2.2 Application Domain

The application domain derives from the rapid digitalization transformations happening in the area of the telecommunication sectors. Telecommunication providers are facing several different aspects of change in technology as well as shifts in business opportunities today. For example, dedicated hardware solutions are moved to the cloud. This phenomenon is described as softwarization, meaning to transform mostly hardware related mechanisms to the virtualized environment through software [34,35]. As described in Chapter 1, this arises with more flexibility, energy advantages, and cost-effectiveness, but also with more complexity and problems regarding the high reliability of the new services (as previous hardware solutions provided 99.999% availability [3]).

This thesis focuses on the two VNF productively applied IMS and video streaming as service use cases for evaluation purposes.

2.2.1 IP multimedia subsystem

The Project Clearwater [1] is considered as one of the first examples for Virtual Network Functions. Project Clearwater is an open-source software, which provides an implementation of the IP multimedia subsystem (IMS). IMS is an emerging architecture for IP-based telecommunication services, such as voice-, video calls or messaging.

The core implementation of Clearwater consists of several microservices (see Figure 2.1), allowing users to register, authenticate, and initiate the connection for calls within the system. There exist the services named Bono, Sprout, Homestead, and Ellis, which are considered within the experiments in this thesis.

- Bono: This service functions as edge proxy, handling client's connections. Clients can register via the SIP protocol in order to initiate calls. The requests are then routed to the Sprout service.
- Sprout: This service manages the different communications to the other internal services, e.g. requesting authentication.

[1] `https://www.projectclearwater.org/`

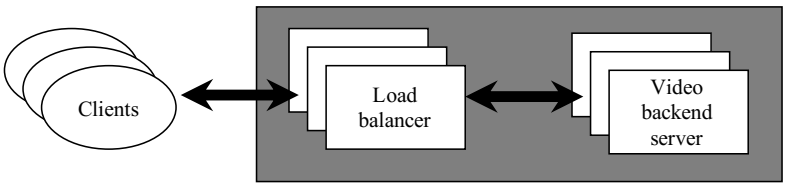


Figure 2.2: Structure of web service components for the video-on-demand service.

- Homestead: This service contains the client profile information, which is needed to authenticate clients.
- Homer: This service contains user-specific service settings documents.
- Ellis: This service obtains the information for the management unit. It functions as an account management system.

Project Clearwater concentrates on the IMS core, such that the focus is the initiating process of calls between users. Calls itself are not handled within the Clearwater service. Clearwater handles mostly user management procedures as well as call management related tasks.

2.2.2 Video on demand

As the increase in network usage is highly influenced by video streaming nowadays [37, 38], the thesis applies a reference implementation of a video-on-demand service.

As described in our work [39], we also used the video-on-demand service as a web service use case in this thesis. The video-on-demand service consists of a setup of three individual web service components (see Figure 2.2):

1. The *video backend server* holds a local copy of the video content and is implemented using the RTMP module[2] of the Nginx service[3].
2. Furthermore, a *loadbalancer* connects clients and replicated backend servers through round robin based DNS balancing.
3. Lastly, there exist user *clients*, hosted on a separated hypervisor. Clients are streaming available video content using the ffmpeg[4] command line tool.

Both service use cases describe the current leading focuses in softwarization of hardware solutions as VNFs as well as network utilization, which the large telecommunication providers need to cope with.

2.3 Analytic Concepts

Resource monitoring data are the aimed data source of this thesis. These are continuously collected and provided to the data analysis component. The monitored data is considered as an incoming data stream. We consider the following properties of a data stream as a definition by Babcock et al. [40].

[2] `https://github.com/arut/nginx-rtmp-module`
[3] `https://nginx.org/`
[4] `https://www.ffmpeg.org/`

Definition 2.3.1 (Data stream [40])

- *The data elements in the stream arrive online.*
- *The system has no control over the order in which data elements arrive to be processed, either within a data stream or across data streams.*
- *Data streams are potentially unbounded in size.*
- *Once an element from a data stream has been processed it is discarded or archived — it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams.*

Furthermore, the monitoring data will be considered within this thesis as time series data stream. A proper definition, which we will use in this thesis, has been defined by Sadik and Gruenwald [23]. Instead of the term time series, they consider the nomenclature of the data stream, which is going to be switched within this thesis to provide consistent names to the reader. Thus, the modified version, which is mainly based on Sadik and Gruenwald [23], is:

Definition 2.3.2 (Time series [23])
A time series is an infinite set of data points $P = \{x_t | 0 \leqslant t\}$*, where a data point* x_t *is a set of attribute values with an explicit or implicit timestamp. Formally, a data point is* $x_t = (v, \tau_t)$ *where* v *is a p-tuple, each value of which corresponds to one attribute, and* τ_t *is the timestamp for the* t*-th data point.*

Albers [41, 42] provided a general definition for online algorithms:

Definition 2.3.3 (Online algorithm [41, 42])
Formally, an online algorithm receives a sequence of requests $\sigma = \sigma(1), ..., \sigma(m)$*. These requests must be served in the order of occurrence. When serving request* $\sigma(t)$*, an online algorithm does not know requests* $\sigma(t')$ *with* $t' > t$*. Serving requests incur cost and the goal is to minimize the total cost paid on the entire request sequence.*

Definition 2.3.3 describes that online algorithms have to provide point-wise analysis properties. Thus, arriving data points have to be computed in the incoming sequence. In order to ensure that each data point can be computed, the computation of the previous data point must be completed. Such demands on the computation time are described as real-time processing and just-in-time processing with respect to the flexibility on the exact time needed for computation. Nelson and Wright [43] described that the *time to decision* is the crucial objective to be considered in such time demanding applications. In this thesis, we distinguish between real-time processing and just-in-time processing as follows.

We consider the term *real-time processing* to provide hard and fixed bounds on computation steps with respect to computation time. By this, the computation must follow an exact number of instructions, which are all bound in time. Thus, the answer of a given analysis step will always provide an answer after a specific amount of time. Such hard constraints are needed in many applications regarding human safety, e.g. assisted surgery, inflation of airbags, ABS systems in cars, etc. Therefore, the *time to decision* is fixed by a defined time.

In contrast to this, *just-in-time processing* provides more flexibility with respect to the *time to decision*. As the name indicates, the computation has to be finished just in time, before a defined scenario happens. For the data stream processing use case, we consider that computation must finish before a new data point arrives. Thus, the *time to decision* is bounded by the monitoring frequency. Within this thesis, we first concentrate on the just-in-time processing definition.

2.3.1 Machine Learning Methodologies

The key indicator for machine learning algorithms is the need of a performance criterion and the need for data to be learned [44]. The performance criterion mostly focuses on the aimed goal of computation like classification, clustering, regression, etc. With respect to the availability of training data for the learning process, machine learning models can be categorized into the following three types: supervised, semi-supervised, unsupervised. Landauer et al. [45], Ramchandran and Sangaiah [22], Hodge and Austin [13], and Goldstein and Uchida [17] all describe these three types for anomaly detection, dependent on the information available for learning, as follows:

- *Supervised learning* requires labeled information for the given data points. All different types of anomalies as well as normal behaving data must be present to the learning approach.
- *Semi-supervised learning* instead consumes a limited subset of labeled information about the normal system state, while abnormal behavior is not provided as labeled information.
- *Unsupervised learning* can cope with non-prior knowledge about labels. In order to determine decisions, this type of learner tries to determine patterns within the given data but also needs further defined requirements in order to distinguish between normality and abnormal data.

Another type of characterization between learners is the number of available training data in time. These are mainly differentiated into offline learning and online learning [46].

- *Offline learning* (also mentioned as batch learning) provides the learning algorithm for the complete dataset. Some machine learning algorithms require the complete set as they are optimized to use random input of training data or compute model updates of the model by multiple repeated runs over such data. To ensure such availability of the complete dataset, the data has to be stored or held in memory, which might not be applicable for many use cases.
- *Online learning* on the other hand does not use the entire dataset at once; it is based on iterative learning of new incoming data points one after another. As a data stream might be endless, online learning is the aimed type of learning method within this thesis.

In the following, we describe different machine learning approaches, which are frequently used within anomaly detection algorithms to model the normal system state and are further considered as baseline approaches for evaluation.

2.3.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [9] is an online clustering algorithm, aiming to cluster data points within big datasets through iterative computation. BIRCH applies the concept of micro-clusters as a key methodology in order to aggregate distribution information of the given data points. This concept enables the possibility of fixed size memory usage and logarithmic time for iterative updates. Thus, BIRCH is efficient in computation also for big datasets but is not considered to be used for time series analysis. The main purpose for BIRCH is the clustering of big datasets, such that data are randomly inserted into the clustering algorithm in order to train an optimized model.

There exist many different clustering algorithms using the concept of micro-clusters in order to summarize data points [9, 47–50]. Micro-clusters are defined as tuple containing three entries: (N, L, S), where

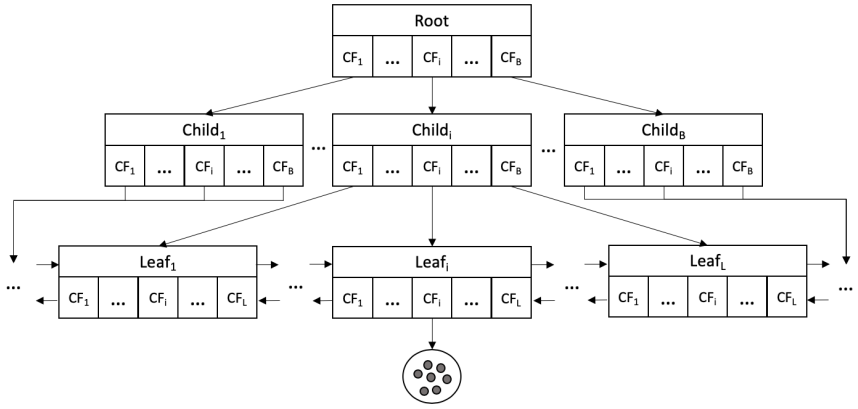


Figure 2.3: Structural overview of a CF-tree.

- N is the number of multi-dimensional data points x_i represented by the micro-cluster,
- L the linear sum $\sum_{i=1}^{N} x_i$ of data points and
- S the squared sum $\sum_{i=1}^{N} x_i^2$.

This notation allows maintaining an aggregated summary of data point sets in an online manner by adding new data points to the three components. Thus, the summary is fixed in memory consumption. Furthermore, clusters can be easily combined by adding the single components.

Furthermore, they proved the additivity property [51,52] to merge micro-clustering tuples by:

$$CF_1 + CF_2 + ... + CF_b = (N_1 + N_2 + ... + N_b, L_1 + L_2 + ... + L_b, S_1 + S_2 + ... + S_b) \quad (2.1)$$

Zhang et al. [9] introduced a hierarchical tree-based model for BIRCH through defining the nodes of the tree storing Clustering Features (CF) within the height balanced CF-tree. CFs internally store such micro-clustering information. Each CF is related to a node within the CF-tree. Furthermore, they use the additivity property 2.1 to define the micro-clustering tuples for parent nodes.

The nodes are stored in the data structure of a height balanced B+-tree, interconnecting leaf nodes with each other as illustrated in Figure 2.3. Figure 2.4 showing a CF-tree containing two CFs in each node on the left side. On the right side, the data points are represented by colored information about the CFs of the leaf nodes. Furthermore, the root node information will be based on the additivity property (see 2.1): $Root.CF_1 = Leaf_1.CF_1 + Leaf_1.CF_2$ and $Root.CF_2 = Leaf_2.CF_1 + Leaf_2.CF_2$ respectively.

Based on the three values stored within the CF, it is possible determining

- a centroid $\frac{L}{N}$ and
- radius $\sqrt{\frac{N \cdot \frac{L}{N}^2 + S - 2 \cdot \frac{L}{N}^2 \cdot L}{N}}$ representing an n-sphere.

Zhang et al. [9] define four phases for BIRCH:

1. *Initialize model by loading data*: This phase represents the training of the CF-tree.
2. (Optional) *Condense model into a desired size*: This phase considers to aggregate the clustering model to be fixed e.g. in height.

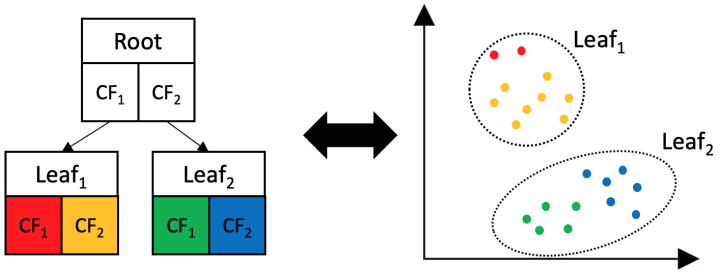


Figure 2.4: Exemplary CF-tree, representing clusters on the right side.

3. *Apply clustering*: BIRCH considers first to build the complete tree and then applying the prediction on the dataset again.
4. (Optional and Offline) *Refine clustering*: Refine clustering: Zhang et al. [9] present the possibility to use offline clustering techniques like k-means when searching a defined number of k clusters. Thus, k-means can be applied to the centroids of the leaf's CFs and therefore be grouped.

The insertion of a new data point can be also described in four phases:

1.1 Identify appropriate leaf node: This phase identifies the closest leaf node by comparing the distance between CFs and the given data point. This can be achieved by applying the Euclidean distance between the centroid and the data point. Another possibility can be to check whether the data point fits into the n-sphere of a CF.

1.2 Modify the leaf node: Given the leaf node, check whether the leaf can absorb the data point without violating a user-defined threshold, which represents the maximum size of a leaf cluster. When the condition is not violated, the point can be added to the corresponding CF. When the condition is violated, the new point is added to a new CF.

1.3 Modify the path to the leaf: As we created a new CF, it is possible that the leaf cannot store that many CFs, so the node needs to be split. Thus, the leaf node is replaced with a new node, which again has new leaf nodes as children.

1.4 Rebuilding: This phase can be applied when multiple splits have already appeared. The goal is to group the closest CFs into the same nodes, which might be separated by previous splits. This improves the overall structure of the tree.

2.3.3 Autoencoder

An autoencoder is a neural network consisting of the two parts encoding η and decoding ζ as illustrated in Figure 2.5. They are used in several different contexts e.g. speech recognition [53], image processing [54] and intrusion detection [55]. Autoencoders are designed to be applied on dimension reduction tasks [56] due to their construction of an internal bottleneck. The encoding function projects the incoming data to a smaller dimension, while the decoding function approximates the encoded version back to the original dimensions. In detail, autoencoders consist of multiple neural network layers, split into $\eta : X \to Y$ and $\zeta : Y \to X$. Basically, the number of nodes decreases to a bottleneck in the encoding, while the number nodes in the decoding section increase again to their original number of dimensions.

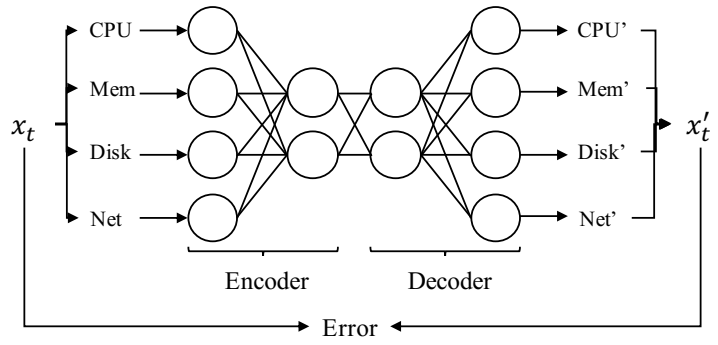


Figure 2.5: Abstract illustration of an autoencoder network with four input and output layers and a bottleneck of two dimensions.

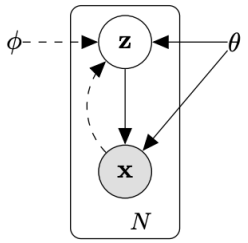


Figure 2.6: VAE model. Source: [58]

Using an autoencoder as an identity function for a given data stream, each data point is learned by simple backpropagation iteratively. Given a data point x, applying both $\eta \circ \zeta : X \rightarrow X$ results in an approximation of the same data point x' for computing the reconstruction error.

Most autoencoders use either simple feedforward neurons or restricted Boltzmann machines as layers.

2.3.4 Variational Autoencoder

In contrast to the defined autoencoder network, there exists an adapted version mapping the encoded values into predefined distributions, called variational autoencoders (VAE) [57, 58]. A VAE is a neural network architecture that is based on the structure of an autoencoder but including distribution models within the bottleneck. The input vector from the encoding network is mapped to a predefined distribution instead of mapping it onto a vector in a traditional feedforward autoencoder. Hereby, the vector of the traditional autoencoder is replaced by two vectors. One of them represents the mean and the other capturing the variance of the predefined distribution. The decoding network is fed through providing such mean and variance from the distribution, which represent the low-dimensional latent space.

Figure 2.6 shows the dependencies of the generative process by illustrating the latent variable z and known variable x, both assume an underlying distribution $\mathcal{N}$ (usually a multivariate Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$). The solid lines describe the generative model $p_\theta(x, z) = p_\theta(x|z)p_\theta(z)$, while the dashed lines approximate the variational value $q_\phi(z|x)$ for the posterior, where $\mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x))$ derive from the neural network.

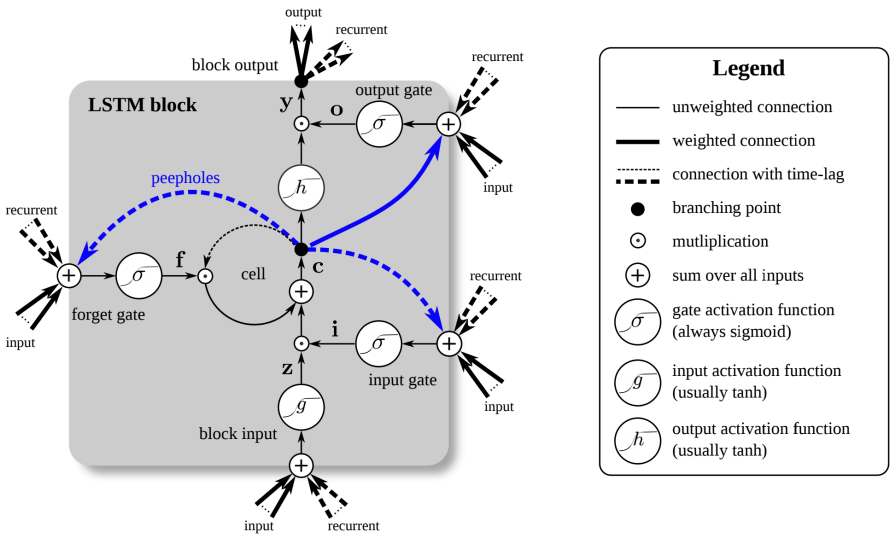


Figure 2.7: LSTM cell visualization. Source: [64], part of the image is taken from the original visualization.

2.3.5 Long Short Term Memory Networks

Traditional feedforward neural networks have no notion of order in time, as they do not consider data from previous iterations. This issue is solved by introducing recurrent loops within the neural network model, making historic decisions from previous iterations available to the current calculation. Such networks are also known as Recurrent Neural Networks (RNNs) [59]. Due to their construction, RNNs capture a fixed number of previous states, so that the concept of Long short-term neural networks gains attention.

In more detail, many RNNs are not able to learn long time lags between relevant input events due to the backpropagated error, which either exponentially increases or decreases, thus effectively getting lost. This vanishing error problem was tackled by introducing several gating units combined as Long Short Term Memory (LSTM) cell [60]. LSTMs are widely used in several different application areas, e.g. stock market price predictions [61], patient diagnosis prediction based on patient's history [62] and video classification [63].

Figure 2.7 by [64] presents a LSTM cell (here called block), consisting of the three gates: input gate, output gate and forget gate. The image also shows the dependencies between the different states and whether there exist weights, computational operators, and usage of activation functions (with the usual usage indication in the legend). Besides the gates, the input and output blocks are shown, which are also present in the RNN context. The incoming value to an LSTM cell is firstly multiplied by the activation of the input gate. The resulting value is further multiplied by the forget gate, which captures the output values of all previous cell values. Lastly, the value is multiplied by the output gate and provided as cell output to the next layers of the network. The gates use element-wise multiplication by sigmoids, so they decide when data is allowed to enter, leave, or be removed. Their own set of weights is also adjusted with the recurrent networks learning process, so over time, the unit learns when to forget information or ignore certain input. So, a state will not impact the network's output. Each LSTM layer consists of a set of these units, which are recurrently connected.

This vanilla version of LSTM cell is also used within this work with the proposed activation functions. The concrete formulas integrated into the framework are described by Griff et al. [64] and defined by Hochreiter and Schmidhuber [60].

2.3.6 Dynamic Threshold Models

Recent works apply dynamic thresholding models in order to differentiate anomaly scores in an online manner [65–67]. A valuable threshold T has to be selected in order to differentiate between normal and abnormal scores.

The following four different models for calculating a threshold are investigated: Gaussian Cumulative Aggregation (CA), Sliding Window Aggregation (SWA), Exponential Moving Model (EMM), and Double Exponential Moving Model (DEMM). All of these methods are well known approaches either calculating statistical values, which are used for building a threshold or are used for smoothing time series data [68].

Gaussian Cumulative Aggregation For the cumulative aggregation based threshold learning, we assume that the reconstruction error Δ is normally distributed, also assumed by [65–67]. Consequently, we define a threshold based on the mean $\mu(\Delta)$ and standard deviation $\sigma(\Delta)$ of the complete historic values of Δ. Based on this assumption, we define a threshold of $T = \mu(\Delta) + c \cdot \sigma(\Delta)$. The parameter c reflects a sensitivity value, as the false alarms can be approximated through T. Obtaining the mean and standard deviation iteratively over time is straightforward and computed in constant time.

Over longer periods of time, new data does not have a great impact on T anymore. This makes this approach difficult to adapt to dynamic and concept changing situations. In order to avoid this behavior, there exist further methods tackling this problem.

Sliding Window Aggregation Sliding windows are a concept of storing the latest historic values. The window has a fixed size, so that for new incoming data, older data points need to be removed. For this purpose, we store the latest reconstruction errors in a sliding window. Based on these errors, the mean and standard deviation are calculated and the same threshold T can be computed as described above. Obtaining these information is also possible iteratively by applying the same mechanism like used in CA with subtracting the older values, which are removed from the window.

A major disadvantage is that an expert has to define the size of the window and may be highly dependent on the current situation of the monitored system. In order to automatically give newer data more impact, the following methods are defined.

Exponential Moving Model The exponential moving model is used for smoothing values or calculating a running moving average and variance. Basically, the latest data has exponentially more impact to the current value than the older data. We use the weighted version of EMM [69], where the user can influence the factor of impact of new data by defining a weight α, also called as learning rate. For a new incoming data point x at time t, the mean μ and variance v can be calculated as:

$$\begin{aligned} \mu_t &= (1-\alpha) \cdot \mu_{t-1} + \alpha \cdot x \\ v_t^2 &= (1-\alpha) \cdot (v_{t-1} + \alpha(x - \mu_{t-1})^2), \end{aligned} \tag{2.2}$$

where $\sigma_t = \sqrt{v_t}$. For the initial configuration we use the first incoming data point as initial μ.

Based on this concept, advanced smoothing models are developed capturing further time series behaviors.

Double Exponential Moving Model The DEMM [68] consists of two functions for the overall trend m of data and the non-trend function, also called baseline n. For a new incoming data point x, the resulting smoothing value x' combines both functions by simply adding the results $x' = m(x) + n(x)$. For DEMM, we use again a weighted model, such that the user can define its own learning rate for both functions, weight α for n, and weight β for m.

Exemplary, we show how the calculations are performed for the mean. The standard deviation can be calculated respectively like shown in EMM.

$$\begin{aligned} n_t &= (1-\alpha) \cdot (n_{t-1} + m_{t-1}) + \alpha \cdot x \\ m_t &= (1-\beta) \cdot m_{t-1} + \beta \cdot (n_t - n_{t-1}) \\ \mu_t &= n_t + m_t \end{aligned} \tag{2.3}$$

.

There exist further advanced models like the Holt-Winters model [70, 71], which additionally includes the seasonality of regularly upcoming changes within the data stream. Here, the user needs to define the exact seasonal length. As we currently do not consider seasonal data, we did not integrate this method to our evaluation and propose this evaluation with other datasets for future work.

2.3.7 Genetic Algorithm

Within this thesis, we concentrate on genetic programming to be used as basis for hyperparameter optimization [72–75]. Genetic programming is also applied in different domains like medical image processing [76–78], feature engineering in stock price prediction [79, 80], electricity load analysis [81] and rainfall forecasting [82, 83].

Algorithm 1 Basic Genetic Algorithm based on [84]

1: *population* ← *random* generation of initial pool of chromosomes
2: **while** *terminationCriterion* is not met **do**
3: Evaluate *fitness* for each chromosome
4: *Select parents* to breed new generation
5: *population* ← *Crossover* parents
6: *Mutate* population

Algorithm 1 shows the general process of a typical genetic program as defined in [84]. It consists of the following main steps:

- *Initial population*: Genes are considered as entities to be optimized (e.g. single hyperparameter). Those genes form a chromosome, which represents the combination of specific genes into an algorithmic solution. The population is a selection of chromosomes and is initiated by this first step.
- *Fitness function*: The fitness function provides a scoring value of which the individual chromosome's performances can be differentiated. This score functions as optimization criterion and for ranking the individuals to distinguish between promising individuals from those, which do not provide good results.
- *Selection*: Based on the fitness scores, the selection of individuals describes which selected individuals will pass their genes to the next generation. These selected individuals are also called parents.
- *Crossover*: When breeding the new generation, we crossover two parents of the set of selected individuals. The crossover describes how the genes of the individual parents are selected to form the child.

		predicted	
		anomaly	normal
actual	anomaly	TP	FN
	normal	FP	TN

Table 2.1: Binary classification possible outcomes with respect to the ground truth value.

- *Mutation*: This step describes whose crossover genes are mutated with some probability. This evolves the population to break out of local optima, thus gaining the possibility to find the global optimum or a better local optimum.
- *Termination*: This phase is optional, but needed when a selection has to terminate. The termination criterion provides the exit case when this optimization technique should stop being performed. For example, when there is no increase detected in the average fitness score, the population has reached an equilibrium.

There exist multiple more complex extensions to optimize the general structure of genetic programming like [85–88]. We refer the interested reader to the survey by Lim [84] providing an extensive overview of different genetic programming adaptations.

2.4 Evaluation Metrics

Evaluations are performed throughout the thesis. In order to provide the reader with a better understanding of the given evaluation metrics, we next define the metrics used.

Pointwise evaluation As anomaly detection is a binary classification task, the output results are expected to be binary (normal or anomaly). Consequently, a positive (anomaly detected) and negative class (normality) can be used for labeling. The assignment of correct and incorrect class labels causes four different possible outcomes regarding the correctness of the prediction compared to the actual state. Table 2.1 illustrates the ground truth values of the given data (*actual*), which can of cause be labeled as one of the both classes normal and anomaly, while the *prediction* also provides both possible outcomes. This results into the four possible states of true positive (TP), true negative (TN), false positive (FP, also type-I error) and false negative (FN, also type-II error). With respect to anomalies, the ground truth of the positive state can be true or false with respect to the prediction, resulting in TP and TN, respectively. The same holds for the negative state representing the normal indicator. Table 2.1 is also known as confusion matrix, where TP, TN, FP, FN represent the counts of appearances. The sum of all these four cells is the number of total observations.

The perfect prediction model contains exclusively counts for true positives and true negatives within the confusion matrix, while false positives and false negatives did not appear at all. In practice, prediction models produce errors and misclassify instances. While the confusion matrix gives an intuition for the quality of the prediction model, there exist further evaluation metrics capturing the quality aspects to compare classification results.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

Equation 2.4 represents an overall score, which describes the proportion of correctly classified instances versus all data points, named accuracy. For imbalanced datasets, this score is also imbalanced [89]. Consequently, TP-rate (sensitivity) and

TN-rate (specificity) can be investigated showing whether abnormality or normality, respectively, are captured more precisely (see Eq. 2.5 and 2.6). Accordingly, FP-rate and FN-rate represent the complement representing the error rates for abnormal and normal behavior (see Eq. 2.7 and 2.8).

$$TP\text{-rate (recall)} = \frac{TP}{TP + FN} \tag{2.5}$$

$$TN\text{-rate} = \frac{TN}{TN + FP} \tag{2.6}$$

$$FP\text{-rate} = \frac{FP}{TN + FP} \tag{2.7}$$

$$FN\text{-rate} = \frac{FN}{TP + FN} \tag{2.8}$$

The recall (TP-rate) covers how much of the predicted anomalies are actual anomalies, while the precision (see Equation 2.9) covers the proportion of anomalies correctly classified in contrast to all anomalies. The F_1 score (see Equation 2.10) represents the harmonic mean of the precision and recall as a combination.

$$precision = \frac{TP}{TP + FP} \tag{2.9}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.10}$$

For imbalanced datasets, which are common for anomaly detection, there exist further metrics representing an overall score like the Matthews Correlation Coefficient [90] and the more conventional evaluation metric the area under curve (AUC) of a receiver operating characteristic (ROC) curve [91]. The ROC curve represents the TP-rate (sensitivity) in contrast to the FP-rate ($1 -$ specificity $= 1 -$ TN-rate). The area under the convex hull of the multipoint-curve represents the AUC value including both type-I as well as type-II errors. The optimum is the area under the simple trapezoid defined by the TP-rate and TN-rate of the model [91]:

$$AUC = \frac{(TP\text{-rate} + TN\text{-rate})}{2} \tag{2.11}$$

The AUC value is also referred to as balanced accuracy as it captures the TN-rate and TP-rate without encountering the dataset's imbalance [89].

Time-based event evaluation The point-wise evaluation metrics do not capture time-dependent metrics. There exist point anomalies, but also anomalies lasting for longer periods of time. Thus, the anomalies appear as continuous sequence of anomalous observations. Let us call such sequences anomaly events, where the self-healing pipeline can try to solve the problem. Anomaly events capture degraded state anomalies and provide consequently the possibility to react.

Figure 2.8 illustrates the key measurements, which are important for arriving at the evaluation metrics. Let us assume that at first there is a normal behavior until the point in time marked by *anomaly injection started*, representing the start of the anomaly event. Identifying the first following observation in time as an anomaly would be the most beneficial for the further analysis and selection of appropriate remediation actions. This time difference, between the start of an anomaly event until it is actually detected (as shown as $\Delta time$ in the image), is investigated for the time-based event evaluation for the different anomaly types.

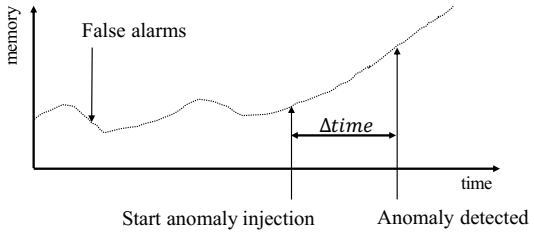


Figure 2.8: Time-based event evaluation principles and key metrics.

As a consequence, we measure the percentage of events, which are correctly detected (TP-rate$_{event}$) and the average (and standard deviation) of the time differences. In addition, the false alarms in the normal signal are measured to provide relevant metrics for production environments. Furthermore, such false alarms may appear as phases. Thus, we also consider the average (and standard deviation) of such false alarms in time as an evaluation metric.

Chapter 3

Related Work

3.1 Characteristics of Service Anomalies

There exist several possibilities grouping anomaly scenarios for services on common properties like severity, impact, or component type or origin of cause [27, 92]. Section 2.1.2 provided a definition of severity based order of anomaly classes. We concentrate on degraded and masked anomalies.

Indication patterns in monitored resource consumption of a system can be grouped into the following categories:

- *Temporal evolution*: The time-dependent change of resources varies in different anomaly scenarios.
- *Signal variance changes*: Resource metrics change in fluctuation over time.
- *Propagation to correlated signals*: Single resource metrics cause propagation to further resource metrics.

With respect to the *temporal evolution*, we distinguish mainly between successive change (increase or decrease) over time or rapid changes. While rapid changes always need a preceding event as a trigger, the former can develop over time without an obvious cause [29]. Such triggering events can be e.g. software bugs or hardware failures leading to instant drops or jumps of the system's resource utilization causing degraded system performance. Jim Grey [93] introduced the terminology of Heisenbugs describing the phenomena of occurring anomalies during productive runtime, which are unable to be detected with modern approaches of automated software testing [29, 94] and bug detection techniques [95]. Grottke and Trivedi [96] described Mandelbugs as a subset of Heisenbugs influencing the execution environment with respect to timing, ordering of inputs, operations, and the time lag between bug activation and failure occurrence [29]. Mandelbugs as well as the generalized Heisenbugs are considered to be examples for triggering events causing rapid changes.

System components can also develop a progressively increasing or decreasing resource utilization over time due to software aging [97] (see Section 2.1.2). For example, Garg et al. [98] evaluated the resource leakage behavior of nine UNIX workstations' operating systems over a total of 53 days and showed the presence of memory leaks and successive increase of file table entries. More recent studies show that software aging is a common part of today's complex and distributed systems infrastructure. The presence of software aging was shown for cloud environments [99], web servers and service oriented applications [100, 101], application server environments [102], and e-commerce applications [101].

Signal variance changes describe the volatility of resource consumption of a system over time. We differentiate between constant changes (small variance) and fluctuating changes (high variance). While rapid changes usually alternate constantly around a value, progressive anomaly scenarios are indicated by a certain trend, which can vary in fluctuation due to its intensity and deviation from the trendline during runtime. Cassidy et al. [103] illustrate the existence of resource fluctuation due to software aging for memory shared database servers and Grottke et al. [104] provide pattern models of resource utilization changes for software aging including variance.

Propagation to correlated signals indicates the instant correlation of a single monitored metric to another resource metric. Zhang et al. [105] showed the existence of correlation of network traffic datasets covering a wide range of link types such as backbone, internal, edge, and wireless. Mestres et al. [106] show for VNF experiment the presence of correlation between network packets and the CPU utilization.

In productive environments, *combinations* of the indication patterns are likely to happen, combining temporal changes in evolution, variance, and propagation to the multivariate resource data. Analyzing such hybrid indicators is complex and time consuming for administrators. Thus, AI-based methods are hoped to help users significantly.

Based on these categorizations, we included resource-based anomaly injections applied to cloud services, providing emulation of degraded state anomalies (see Section 7.1.2).

3.2 Anomaly Detection

Supervised Anomaly Detection General classification algorithms can be utilized for anomaly detection as anomaly detection is a binary classification problem. A collection of classification algorithms are available by the WEKA machine learning framework [107], which are presented in the following.

J48 is based on the implementation of the C4.5 decision tree learning algorithm by Quinlan [108]. The C4.5 algorithm constructs a decision tree by splitting up nodes on the metric with the highest normalized information gain. Starting from the root node (first node in the tree), the tree is split up until the information gain falls below a predefined threshold. In an optional pruning phase, the resulting tree is optimized from the leaf nodes upwards to decrease overfitting effects. J48 is applied in many different anomaly detection scenarios like network intrusion detection [109, 110], breast cancer detection [111], credit card fraud [112].

Logistic Model Tree (LMT) [113] uses the C4.5 decision tree splitting criterion, but works with logistic regression models in the tree nodes, instead of the constant values used in simple decision trees. The LogitBoost algorithm [114] is used to efficiently compute the regression models for the internal decision nodes. A point to be analyzed is classified by traversing the tree from the root node and testing the input sample against each contained regression model to decide which path should be followed down the tree. LMT is recently applied to e.g. Intrusion detection [115] and anomaly detection for industrial control systems [116].

Hoeffding Tree is an online decision tree learning algorithm, which was proposed by Hulten et al. [117]. As the online scenario accepts a potentially endless data stream to learn, the authors use the Hoeffding bound [118] to predict the number of data points needed to be learned for a new decision criterion in order to approximate the deviation of the unbounded data stream for single features. Taking the number of samples into account, the Hoeffding Tree algorithm operates on a window of data. The splitting criterion is the information gain, as used in the C4.5 algorithm. Hoeffding Trees are applied also for intrusion detection [119, 120] and surveys show the applicability of Hoeffding Trees and adapted versions of them for anomaly detection [121, 122].

The Random Tree classifier constructs a decision tree by randomly choosing the features evaluated for each decision node [123]. The Random Forests classifier [123] extends the Random Tree algorithm by applying a multitude of Random Tree models, each trained with a random subset of the total of available features. Random Tree and Random Forest are applied recently in anomaly detection tasks like credit card fraud [112], insurance fraud [124], database query access [125].

Naive Bayes [126] learns a Bayesian network as an underlying model. A Bayesian network consists of a graph-based model indicating dependencies between values with associated conditional probabilistic tables. Naive Bayes uses a strong conditional independence assumption, which states that all attributes of a data point are independent, in order to efficiently learn the model. The applicability of applying Naive Bayes to anomaly detection tasks was shown by Sebyala et al. [127] and Muda et al. [128] for intrusion detection as well as for breast cancer [129] and lung cancer detection [130].

Further classification algorithms, which WEKA provides, are summarized as follows. The Reduced Error Pruning Tree (Rep Tree) applies additional reduced-error pruning [131] as a post-processing step of the C4.5 algorithm. The Decision Stump classification algorithm is based on the C4.5 decision tree learning algorithm but uses only one layer for decisions [132]. The algorithm stops learning when the best feature with the most information gain is added. Decision Table maps rules to learned classes [133]. In general, decision tables divide up the large hypothesis space to smaller areas, where an optimal feature subset is chosen for the decision table representation. JRip is a classification algorithm learning a set of rules and was proposed by Cohen [134]. JRip uses incremental pruning to reduce the error by finding core rules describing a class to be learned. ONER [135] is a rule-based classification algorithm using minimum-error attributes for prediction. Frank and Witten [136] proposed the PART algorithm, which utilizes lists of decision rules as a prediction model. For each class, a C4.5 decision tree is created and leaves represent a rule for the class. Sequential Minimal Optimization (SMO) [137] is an optimized version of Support Vector Machines (SVMs). As SVMs need to solve large quadratic programming problems, SMO divides it into smaller quadratic programming subproblems in order to efficiently learn large sets of data.

Saibharath and Geethakumari [138] propose an architecture for anomaly detection in cloud environments through applied log analysis for VMs, network behavior metrics etc., applying WEKA-based the described classification approaches. Chavan et al. [139] and Krishna and Rao [140] investigated malware detection utilizing the classifiers Random Forest Random Tree, LMT, J48, SVM.

Rezvani [141] proposed a framework to evaluate machine learning algorithms in a simulation environment, running different cloud-based services and security-related attacks. Rezvani applied J48, Naive Bayes, Decision Table, and Random Forest on various intrusion detection tasks and showed their applicability. Garcia-Teodoro et al. [142] provided a survey for anomaly-based intrusion detection systems including the description of reviewed supervised machine learning approaches like Bayesian Networks, neural networks, and statistical approaches to classify types of attacks.

Sauvanaud et al. [143] applies the supervised Random Forest [123] approach on individual service-components. Through aggregation by weighted voting, the ensembler predicts anomalies for the overall service. The evaluation is based on the VNF scenario Clearwater likewise to our setup as presented in this work. The results show the applicability of supervised approaches in order to detect degraded state anomalies.

Adamova et al. [144] evaluate anomaly detection approaches to a combination of network attacks on service migration in cloud environments. They show that Expectation-Maximization (EM) clustering applied as a supervised technique provides high AUC values. Clusters are generated for individual normal and abnormal behaviors. Huang et al. [145] also investigate anomaly detection approaches for service migrations in virtualized environments by applying the clustering techniques EM-

clustering and proposed an adapted version of LOF with abnormal feature reasoning.

Semi-supervised Anomaly Detection For anomaly detection in time series, a lot of works apply statistical methods to analyze the underlying behavior and predict deviations of its statistical distribution. Statistical parameters like k-sigma [146, 147], statistical hypotheses testing [148, 149], Multivariate Adaptive Statistical Filtering [150], Holt-winter method [151], exponentially smoothing [152], window based PCA [153], robust PCA [154, 155], relative entropy [156], Kolmogorov-Smirnov test [157] are applied to find anomalies.

One-class Support Vector Machines (OSVM) is trained with normal training data [158]. It creates decision boundaries (frontier) through hyperplanes as a kind of convex hulls around the normal data in order to differentiate the abnormal data points. OSVM utilizes a kernel (e.g. RBF kernel) and a scalar parameter to define the frontier. The margin of the OSVM is configurable as a parameter corresponding to the probability of finding abnormal observations outside the frontier.

Isolation Forest [159] utilizes the mechanisms of Random Forests to create multiple random trees through selecting features and the split value randomly. As an assumption, the algorithm assumes the knowledge about the maximal and minimal values for selected features, while Random Forest applies splitting conditions like information gain. The tree is built up until samples are isolated. The path length from root to the isolated sample is measured and reported as anomaly score and averaged over the multiple trees. Abnormal samples are indicated with a short path length compared to normal data points.

Local Outlier Factor (LOF) [160] measures the local density of data points with respect to its neighbors. The LOF score reflects the ratio of the average local density of a data point's k-nearest neighbors and its own local density. Abnormal data points are expected to have a smaller local density in contrast to its k-nearest neighbors, while normal data points should be represented by similar local density.

Gaussian Mixture Models (GMM) [161] are able to build clusters from trained normal data. Each cluster is represented through a Gaussian probabilistic density function. Thus, probabilistic regions can be defined where normal data are expected, while abnormal points are expected in regions with low probability. Reddy et al. [162] show the applicability of GMM for anomaly detection of security-related issues in network traffic, likewise to Bahrololum and Khaleghi [163] providing results to intrusion detection.

The BIRCH [52] (see Section 2.3.2) clustering algorithm was also proposed for anomaly detection in various papers for money laundry detection [164] and KDD'99 datasets [165]. Normal data is fitted to clusters by BIRCH, providing border definitions through the micro-clusters in order to differentiate abnormal data, which is assumed to be outside such micro-clusters [165].

The applicability of neural networks are investigated for anomaly detection in recent years (e.g. surveys by Chalapathy and Chawla [166] and Chandola et al. [18]). Autoencoders (see Section 2.3.3) are for example recently applied to anomaly detection tasks like credit card fraud detection [167], malware detection [168], intrusion detection [55], and anomaly detection for indoor wireless sensor network as IoT-application [169]. Through training normal data, the Autoencoder is able to capture the behavior of the normal signal resulting in a low reconstruction error. By user-defined thresholds or statistical analysis, differentiation of normal data and abnormal data is possible as abnormal data is expected to have higher reconstruction values. Anomaly detection through VAE networks (see Section 2.3.4) can be utilized in the same way as Autoencoders and is applied for example for mechanical vibration signals of a motor [170]. LSTM-based neural networks (see Section 2.3.5) are as well applied for anomaly detection in automotive systems [171] and medical ECG signal analysis [172], which also show the advantages for time-series analysis. AELSTM combines

LSTM cells within an Autoencoder neural network structure. These hybrid models are aimed to provide the capability to add time-based information to the Autoencoder structure, which has no notion of time modeled. An example of an anomaly detector is the recent usage for detecting abnormal behavior of robot-assisted feeding [173].

Self-organizing maps (SOM) are a type of neural network, which provides competing features due to its network topology. Thus, features can distinguish to be different in importance to map a given e.g. data stream. This behavior is for example applied to intrusion detection [119] and for anomaly detection for IT-system monitoring [174]. For the latter case, the SOM is configured to represent the overall IT-infrastructure in order to predict the overall performance of the system. Liu et al. [174] show that anomalies within virtual machines can be detected in related regions of the infrastructure. In contrast to our work, we do not aggregate data from several hosts in a single model but create models per individual component in order to capture its own behavior.

Unsupervised Anomaly Detection Unsupervised anomaly detection does not assume the existence of any labeled data. The key difference in semi-supervised learning is the absence of normal labels.

Laptev et al. [147] developed a framework for unsupervised anomaly detection for univariate time-series data called EGADS. EGADS includes several different anomaly detection approaches. The first group of anomaly detection techniques learns a reconstruction of the time series (e.g. Arima, Exponential smoothing), where the prediction error (Euclidean distance) or the relative error $\frac{x_t}{u_t}$ is applied and a threshold selected through modeling these errors to a Gaussian distribution and $T = \mu + 3 \cdot \sigma$. The second class of anomaly detection techniques decompose the time-series into trend, seasonality, and noise. Whenever the noise component exceeds a threshold defined limit, the data point is reported as anomaly. The threshold is again defined by the threshold function described above. The open source implementation[1] assumes the existence of the complete dataset although the methods can be partially applied in an online manner.

Hundman et al. [65] provided an unsupervised anomaly detection based on LSTM and dynamic thresholding for data streams. From a multivariate time series, a sequence of a fixed number of data points is selected and feed into the model. The presented model applies an LSTM neural network for every single dimension and predicts the following data point. Based on the prediction, the prediction error is calculated by dimension-wise euclidean distance. They apply exponentially-weighted moving average to smooth the errors. Based on the smoothed errors, the threshold is calculated, which is based on the mean and standard deviation of the set of smoothed historic errors by applying the Gaussian Tail. Furthermore, they propose mitigating false positives through postprocessing and reclassification of data points by selecting a sequence of errors, which produce the maximum smoothed errors but are labeled as normal. Another approach would be to investigate the severity of the error and select a border. Experiments showed that for two NASA datasets the approach can be applied to telemetry data and provided a total recall of 90.3% for point anomalies and 69% recall for contextual anomalies.

Malhotra et al. [66] also proposed anomaly detection through LSTMs. The model is trained based on assumed normal data points based on a fully connected neural network with two layers of recurrently connected LSTM cells. The model forecasts the next l data points and computes an error based on the Euclidean distance of every single dimension. Such vectors of errors are used to fit a multivariate Gaussian distribution. Thus, for new arriving data points, a probability of normality can be predicted. Based on a user-defined probability threshold, abnormal data points can be detected.

[1] `https://github.com/yahoo/egads`

For this study, the approach estimates this threshold based on an anomaly validation set. The experiments were executed on four different datasets among ECG signal analysis, Space shuttle data, engine sensor data, and power consumption dataset. The results show a high precision ($> 71\% - 98\%$), but low recall ($< 20\%$).

Oh and Yun [67] investigated anomaly detection for machine sound data. Here, the sound signal is transformed through Short-time Fourier transform (STFT) [175] into the frequency domain. The resulting spectrogram is further used to train a feedforward autoencoder utilizing a U-net compression structure of hidden layers. Likewise to Malhotra et al. [66], the reconstruction error is used to train a Gaussian distribution representing the normal error behavior and to determine a fixed threshold based on a validation set. The article shows the applicability for the machine sound use case with more than 90% accuracy.

Xu et al. [176] propose the approach Donut for detecting abnormal behavior in seasonal web application KPIs. The proposed approach utilizes a sliding window of user-defined KPI observations and applies a variational autoencoder (VAE) to learn the reconstruction of observations in a window. In contrast to the feedforward variant of an autoencoder, a VAE encoded values into predefined distributions [57, 58]. Furthermore, Donut uses the reconstruction probability [177] which includes, beside the reconstruction error, also the variance parameter of the distribution function. They stated the applicability of the approach by analyzing KPIs data of a top global internet company achieving F-scores from 0.75 to 0.9.

Ibidunmoye et al. [178] also analyze the performance of IT-services, but also infrastructures, in order to detect anomalies. The two unsupervised methods Behavior-based anomaly detection (BAD) and Prediction-based anomaly detection (PAD) are presented. BAD analyses statistical properties of a tumbling window by applying kernel density estimation [179]. Anomalies are predicted whenever the density estimation deviates from the low side of the Shewharts control chart [180] compared to the latest tumbling window. PAD applies forecasting based on the knowledge of the latest data points in a sliding window. A cubic spline model is applied to forecast the next data point. Based on the forecast and the actual observed data point, a prediction error can be calculated. Again an adaptive threshold is applied to recognize anomalies based on the errors. As a threshold, an exponentially weighted moving average control chart [180] is applied to the errors of the recent sliding window. Based on the threshold, provided by the control chart model, anomalies are triggered whenever the prediction error exceeds these bounds.

Ahmad et al. [181] propose an anomaly detection based on Hierarchical Temporal Memory (HTM) networks, which is a special type of artificial neural networks inspired by sequential learning and sparse activity in the human cortical regions (thus does not use fully connected layers, but sparse connections between nodes). HTMs are a rather new sequence learning approach improving with modern neuroscience [182] compared to longstanding LSTMs, which have been researched for nearly two decades [60]. The HTM network's core components consist at first of an encoder and a spatial pooler creating a sparse representation of an incoming sequence. The resulting sparse binary vector is used as one of the two outputs of the HTM network. The second output is produced by the sequence memory cell of the HTM network reconstructing the sparse input. Based on these two outputs the prediction error is calculated and an anomaly alert is triggered whenever the prediction error exceeds a threshold based on the mean and standard deviation of an assumed Gaussian distribution of the error and a user-defined sensitivity parameter.

Cotroneo et al. [183] use statistical correlation analysis between system components in order to automatically detect anomalies. Changes within these correlations are predicted as anomalies. The focus of Cotroneo et al. [183] is to combine data from different hosts. This approach suffers from scalability when applied to dynamic service and infrastructure environments. Thus, we aim to create detection models for

individual hosts, which can be applied directly to the monitored entity.

Dean et al. [184] also utilize SOMs, but propose an unsupervised detection model. They aim to predict failures within a cloud infrastructure, but do not consider degraded state anomalies as a use case.

3.3 Concept Adapting Clustering

Chapter 5 develops a concept adapting online clustering approach based on BIRCH [9, 52]. The corresponding related research is presented in the following. It consist of an overview of different online clustering approaches and specific BIRCH variations. As we provide a concept adaptable method, we also present decay methods, which are applied to online clustering approaches.

Online Clustering BIRCH [9,52] is an agglomerative hierarchical clusterer storing summarized cluster information as cluster features in a balanced tree structure. The cluster features consist of a 3-tuple, called micro-cluster, capturing hyper-spherical shapes represented by a centroid and radius. BIRCH provides iterative inserts of new data points and therefore provides iterative updates of the model. Consequently, the model can be applied in an online manner. Through rebalancing, cluster features are able to be merged due to the additive property of micro-clusters [9]. The approach assumes randomized input of a large, finite dataset and a fixed number of searched clusters. The resulting centroids aim to represent the centroids provided by k-mean clustering, but with efficient iterative computation by a fixed-sized tree data structure.

Likewise to BIRCH, scalable k-means [185] summarizes randomly chosen data points from a large, finite dataset by a set of clustering features. The memory consumption is continuously checked and clustering features are summarized when the buffer size of the memory is exceeded. K-means clustering is applied to the compressed information and terminated e.g. when centroids are not moving anymore.

Farnstrom et al. [186] analyzed the proposed compression techniques by Bradley et al. [185] and suggested a simplified variant of a single-pass k-means algorithm. They showed that the simplified variant outperforms any compression variant proposed by Bradley et al., but still provided same qualitative results.

Ackermann et al. [187] developed the StreamKM++ clustering algorithm, which combines coreset as data stream summary and k-means++ [188]. Like BIRCH, a tree structure is built, but instead of using micro-clusters, coresets are utilized. Ackermann et al. [187] showed that the runtime is significantly slower than BIRCH, but clustering results outperform those of BIRCH.

Aggarwal et al. [189] investigated the problem of the influence of historic data of single-pass summaries applied by online clustering approaches and proposed a time window based approach, named CluStream. The time windows aim to capture different lengths in historic time frames and therefore significant changes can be employed. Different length of time windows are captured in a pyramidal structure with micro-clusters. Kranen et al. [190] handles time frame summaries in a tree structure, called ClusTree. The tree structure includes a self-adaptive index structure of micro-clusters. Zhou et al. [191] give more recent data points higher impact on a cluster feature data stream summary. They developed the SWClustering algorithm applying an exponential histogram of cluster features through sliding windows.

The online clustering approaches, described above, are mainly inspired by k-means and provide a technique for summarizing the data stream. Consequently, such algorithms are not capable of detecting arbitrary shapes as they use e.g. micro-clusters, which provide spherical shapes. Density-based clustering like DBSCAN is therefore applied in order to detect arbitrary shapes as they assume neighborhood dependencies through e.g. Euclidean distance but not structural properties. Chen and Tu [192]

proposed D-Stream, a density-based online clustering approach. D-Stream applies density grid mapping in an online manner while applying offline density clustering as a secondary process on the resulting finite grid structure. Cao et al. [49] presented DenStream, which applies data stream compression through micro-clusters, which represent core-points of the density structure. Additionally, they included a novel pruning strategy to grow new emerging micro-clusters, while avoiding outliers.

Further online clustering techniques for multivariate time series data were investigated by Rodrigues et al. [193] maintaining a hierarchical tree structure of clusters. By applying correlation-based dissimilarity measures, each node in the tree splits the time series into the farthest neighbor pairs. Gama et al. [194] apply a distributed stream clustering approach to a sensor network. It combines distribution mapping for state representation, which are reported to a centralized component managing the distribution mappings globally and can apply updates locally at the sensors.

BIRCH Variations Lorbeer et al. [195] introduced A-BIRCH. A-BIRCH provides a solution for automatic estimation of an optimal threshold through Gap Statistics [196]. In addition to the radius, the threshold decides whether nearby points of a cluster are integrated into such. Thus, the threshold influences the structure of micro-clusters. In the original paper of BIRCH, Zhang et al. [9] proposed using linear regression based on the enlargement of the cluster radii of the tree as an approximation of the threshold.

Lorbeer et al. [195] additionally introduced the Multiple branch descent BIRCH (MBD-BIRCH). MBD-BIRCH adds a mechanism to BIRCH, which assigns for a new arriving data point multiple decent leaf clusters beside the closest by distance. They propose to search for nearby clusters in a user-defined distance s around the closest cluster's centroid. Lorbeer et al. [195] showed, that the higher s, the less splitting is applied in BIRCH, but it will also slow down the algorithm as the number of branches to descend increases.

Fichtenberger et al. [47] describes a modification to BIRCH, which provides a provable approximation of the trained cluster centroids compared to k-means centroids. The modified version is referred to as BICO as it includes coresets into the tree structure of BIRCH instead of using the original micro-clusters within a cluster feature. Fichtenberger et al. [47] showed that the quality of results is higher compared to BIRCH due to the provable bounded guarantee.

Guo et al. [197] proposed LBIRCH aiming to provide better results for non-spherical cluster structures. They include a neighbor table (based on the ROCK algorithm [198]) of data points, which does not fit directly any micro-cluster at first. Based on the table, the merging of micro-clusters is identified. Guo et al. [197] showed better results of arbitrarily shaped datasets compared to BIRCH.

Li et al. [199] provide a method to overcome the problem of arbitrary shape clustering. They combine the fast computation of BIRCH with the Artificial Immune Network Clustering algorithm (aiNet) [200]. At first, BIRCH is applied to summarize data into the Cluster feature tree data structure. Afterwards, aiNet utilizes the micro-clusters to learn arbitrary shaped clusters.

Concept Adapting in Online Clustering Being robust against concept changes can be applied through different methods. SWClustering [191] keeps the recent data as histograms in sliding windows. CluStream [189] applies a pyramidal structure in order to capture different time windows.

Besides window-based separation of data points in time, Cao et al. [49] apply a fading function based on exponential decay of $f(t) = 2^{-\lambda \cdot t}$, where $\lambda > 0$ represents the rate of decay and t a successive increasing time value. They propose that the user should define t, thus λ is calculated and adapted depending on the specified t. Chen and Tu [192] also utilize an exponential decay, but applied on the grid density

coefficients just when updates are applied in case of newly arriving data points. Consequently, the exponential decay is defined by λ^{t-t_c}, where $\lambda \in (0,1)$, t defines the time of the last update of density grid coefficient and t_c represents the current time of the newly arriving data point.

3.4 Hyperparameter Optimization

Hyperparameter optimization (tuning) aims to find suitable hyperparameters for a given algorithm in order to perform at its best. In this work (see Chapter 6), we concentrate on determining hyperparameters for the generalization of unsupervised anomaly detection algorithms as defined in Chapter 5.

In general, changing the configuration of the hyperparameters of a given machine learning algorithm also changes the functionality of the algorithm. For example, selecting the aimed number of searched clusters k in k-means forces a different categorization of data points. When such parameters cannot be determined by experts, they need to be decided by another process. Such selection processes can be performed automatically, but assume the existence of an optimization criterion (also called objective). For anomaly detection, one can choose as objective e.g. accuracy, TN-rate, TP-rate, or similar qualitative metrics. Thus, the automatic process can try out a wide range of different sets of hyperparameters and is able to rank those by its objective. This type of hyperparameter optimization strategy is referred to as random search. Random search evaluates a finite set of randomly chosen hyperparameters and consequently might not be able to find the global optimum unless the finite set captures the complete set of possible configuration options. Here, all different possible combinations are tested in order to determine the global optimum, which is also referred to as exhaustive search. Of course, this comes with the cost of time complexity when aiming to search the complete space [201].

Optimization Strategies With respect to the strategy of search (optimization strategy), there exist several approaches, which apply greedy methods to the selection process in order to avoid the evaluation of all possibilities.

Simulated annealing [202, 203] applies a statistical model based on an energy function (e.g. Metropolis equation [204]) to model the probability to change hyperparameters in an improving direction. Depict changes in parameters, which improve the objective, are always selected and new options are evaluated through iterative hill-climbing. The best performing options are reported in the end. Simulated annealing is investigated in several different domains like chemical crystal structure predictions [205], curriculum course planning [206], energy production optimization in water supply networks [207], and performance improvement in learning conversational neural networks [208].

Sequential model-based optimization (SMBO) [209] learns the past experience by a surrogate model to approximate the expected objective. SMBO conducted with Gaussian processes as surrogate model is called Bayesian optimization [210–212] and is one of the most popular SMBO approaches.

F-race algorithms [213] incorporate also computational costs whether it is appropriate to carry on with the search in order to find better parameters. The concept of racing was first applied to solve model selection for machine learning algorithms [214, 215] and generalized as F-race by Birattari et al. [213] incorporating the Friedman's two-way analysis of variance by ranks [216]. A practical implementation is provided with the irace package providing automatic hyperparameter configuration for the state of the art machine learning algorithms [217].

Particle swarm optimization (PSO) [218] is inspired by the social behavior of bird and fish colonies. Such groups of social organisms share information among the popu-

lation by interaction, which is emulated by the PSO [219]. The swarm describes hereby the population's decisions instead of individual optimization. All individual participants in a swarm optimize their own objective but contribute to the overall objective through information exchange. Individuals capture their position and velocity and try to optimize their individual objective, while such individual outcomes decide together on the movement of the whole swarm to a new relative location [220]. Suggestions for the particular implementation options are provided in the empirical study [221] and survey [222, 223]. Poli et al. [223] provide an overview of the different application areas including investment portfolio selection [224, 225], motor control in electric and hybrid vehicles [226, 227], cancer classification [228].

Inspired by genetic concepts of genes, mutations, and crossovers, genetic algorithms [229] also use a population-based formulation for optimizing an objective. A random set of start population is used to rank the best performing individuals. A parent set is selected among the best individuals to grow a new population through a crossover. Additionally, mutations are applied to model randomized changes to enable the option to break out of local optima in order to find the global optimum. Genetic optimization is also applied in different domains like medical image processing [76–78], feature engineering in stock price prediction [79, 80] and medical SVM-based classification [230], electricity load analysis [81], and rainfall forecasting [82, 83].

Hyperparameter Optimization for Machine Learning Any of the above presented optimization strategies can be used for the task of hyperparameter tuning when defining suitable objective functions. For example, the F-race algorithms [213] were developed with the aimed task of hyperparameter optimization, while the other optimization approaches were developed for several different purposes and later applied to the task of hyperparameter optimization.

Connolly et al. [231] showed the applicability of PSO for ensembler ranking of machine learning models for video-based face recognition. The ranking procedure can also be used for hyperparameter selection where the ranking score describes such models, which are the best to be applied. Meissner et al. [232] and Lorenzo et al. [233] show the applicability of hyperparameter selection for configuring deep neural network architectures utilizing PSO. Furthermore, Lin et al. [234] applies PSO for the problem of feature selection and hyperparameter selection for support vector machines.

Fischetti and Stringher [235] combined stochastic gradient descent with simulated annealing for hyperparameter tuning of deep neural networks. Tsai et al. [72] focuses also on hyperparameter tuning of deep neural networks but using a hybrid Taguchi-genetic algorithm, while Nalçakan and Ensari [236] showed the applicability using a general form of a genetic algorithm tuning the number of hidden layers, number of neurons in hidden layers, and activation function. A non-traditional neural network approach is called Hierarchical Temporal Memory (HTM) for unsupervised anomaly detection in data streams. The continuously learning HTM network is reported to be less sensitive to changes in its hyperparameters [181], but may need to be analyzed in future work. A survey of neural network structure tuning approaches is provided by Elsken et al. [237] including additionally advanced inter- and extrapolation strategies for speedup mechanisms to find optimized objectives. Domhan et al. [238] extrapolate learning curves as a performance estimation strategy to identify and discard slowly learning models. Even more advanced, Liu et al. [239] train a surrogate model that scores the performance of the neural network structure in order to forecast performance behavior for deep convolutional neural networks.

Kiss et al. [240] propose a framework for hyperparameter tuning of black-box machine learning algorithms. Their focus is to provide an experimental and understandable UI for educational purposes. The black-box approach utilizes several different of the above described search approaches and optimizes the objective of accuracy through e.g. random search, genetic algorithms, PSO, simulated annealing. They additionally

included fish school search [241], differential evolution [242] and SMBO with Gaussian processes (Bayesian optimization) [210]. The user is then allowed to select upon the different approaches.

Hyperopt [243] is a python library providing hyperparameter optimization for black-box machine learning models of the Scikit-learn library [244]. Bergstra et al. [243] showed the applicability of Hyperopt utilizing Bayesian optimization as search strategy.

Likewise to Hyperopt, Optuna [245] is a python library for hyperparameter optimization for black-box machine learning models, but enables the usage of several different search strategies (e.g. evolutional optimization through covariance matrix adaptation [246] and Bayesian optimization) as well as including performance strategies (e.g. automated early stopping [247, 248]).

Emerging frameworks provide automation for deciding on hyperparameter tuning optimization tools and include as well feature selection, called AutoML [249–251]. Such AutoML frameworks apply also black-box machine learning model optimization for classification tasks. Thus, they expect a labeled dataset capturing the ground truth and normally run 10-fold cross-validation on the given dataset and use accuracy as their objective. As main optimization approaches, Kotthoff et al. [249] and Feurer et al. [250] apply Bayesian optimization, while Olson and Moore [251] utilize genetic programming. Gijsbers et al. [252] evaluated these AutoML frameworks on 39 datasets for solving classification problems. The results show, that the different frameworks result in a significant difference in accuracy. The explanation of the difference for the applied optimization strategies with respect to given data properties have to be investigated in future work [252]. Tu et al. [253] introduced AutoNE, an AutoML framework specialized for network embedding machine learning approaches.

Li and Malik [254] propose a meta-learning in order to learn the optimization of algorithms through reinforcement learning. They showed the applicability for neural network learning [255].

The above described state-of-the art hyperparameter optimization approaches train and optimize the models offline [249,256]. For conducting the offline trained models in a data stream setting, the training data requires to be stored in memory. Furthermore, the optimization process is applied as batch processing combined with cross-validating for ranking purposes [257], which requires at first a lot of time, resources, and labeled data. Researchers acknowledge the demand for automatic hyperparameter optimization for unsupervised models due to the problem of offline hyperparameter optimization [181,258], but the previously described anomaly detection approaches rely on some assumptions about the data distributions. However, it remains impractical to apply for online arriving monitoring data. Thus, we aim to provide a solution, which can be applied in an online scenario and applies a scoring function as objective, which can be used with semi-supervised anomaly detection approaches. For supervised anomaly detection, we assume that one applies the above described approaches as labeled data are present.

Chapter 4

Framework for AI-based Anomaly Detection

4.1 ZerOps Framework

In order to provide an overall self-healing analysis pipeline of multiple AI analysis steps and the proposed scope and placement of the anomaly detection, we propose the ZerOps framework. In [8], we defined the general structure of such an AIOps platform with a closed-loop, which is presented in Figure 4.1. In the middle part (in green), the image shows a typical cross-layer infrastructure setup consisting of (from bottom to top) physical resources on which a hypervisor is deployed, managing virtual resources (e.g. virtual machines), in which services (e.g. VNFs) are deployed. On all different layers, we assume that monitoring data is collected in a stream-wise manner, which sends the data to the self-healing pipeline.

At first, the self-healing pipeline transforms the incoming data into the same data format in the *Monitoring data sink*. In order to do so, information about the system architecture is assumed to aggregate information as well as provide information to the analytics pipeline regarding e.g. normalization.

The data is being sent as a data stream to the self-healing analysis steps, which run all necessary analytics for gaining insights about the system state. This *Data analysis* can include further system knowledge about faults, which can be integrated with e.g. a fault-catalog. As we like to provide a zero-touch solution, this information may also not be included. Thus, our data analysis should be aware of also operating without this prior knowledge. Furthermore, the data analysis triggers events when anomalies are detected and aggregates all needed information to the event, which are needed for the *Self-stabilization engine*, which is also called the Remediation engine later in the thesis. This Self-stabilization engine selects and plans the executions of stabilization actions (also referred to as remediation actions) through either cloud management interfaces like OpenStack API for e.g. orchestration purposes or directly executes commands within the different layers of the IT infrastructure. Of course, this depends on the access rights and the scope on which this architecture operates. The proposed Figure 4.1 assumes a private cloud use case, where all layers can be accessed and operated by the user of the AIOps platform. Through the execution of such stabilization actions, the system is going to be continuously monitored, and hopefully, the anomalies will be remediated. Otherwise, when actions do not have the aimed impact (anomalies are still triggered), the self-healing system is capable of executing further actions, which might be of a higher level of risk. Thus, the system provides a closed loop for zero-touch administrating IT-infrastructure and provides high reliability to the system.

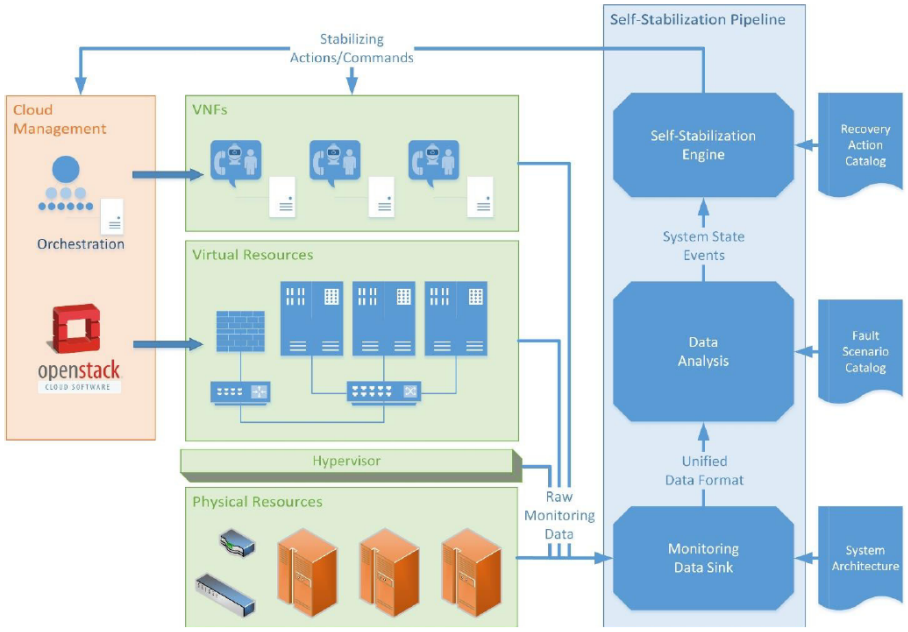


Figure 4.1: AIOps platform for zero-touch administrations. Source: [8]

Figure 4.2 provides more insights about the aimed components of the holistic framework, named ZerOps. On the left (in red), the framework assumes, that there exist virtual resource managers (for this thesis we built a testbed using OpenStack, Docker, and Kubernetes) as well as monitoring services collecting data as a data stream. These can be gathered by any external monitoring tool, but for simplicity, we assume to collect data using the Bitflow collector [259]. Section 7.1.1 provides further information about the metrics collected by the Bitflow collector.

These collected monitoring data are sent to the *Data ingestion*, while information about the virtual resources is collected within a *Topology discovery* service, which provides a dependency model to the data analysis. Given these different data information, all are sent to the *Data analyses orchestrator*, which manages and orchestrates the self-healing analysis pipelines. Those are needed, as the platform also aims to provide decentralized execution of analysis components, for example for the anomaly detection. Also, the orchestration is capable of automatic deployments of analysis pipelines when e.g. services are newly deployed. The data analysis orchestrator also manages which data is sent to which analysis component as not all analysis steps need information about e.g. the topology. The self-healing analysis components are divided into three main parts. First, for each monitored component an anomaly detection analysis is applied. When anomalies are detected, this analysis component sends events to the root cause analysis, which gathers further topology information to determine the concrete root cause of the problem as anomalies may propagate through many different components, which then can be correlated. Based on this information about the root cause and propagated abnormal components, the decision engine aims to determine suitable remediation actions and has to plan its executions into the system. In order to close the loop, an execution engine provides an automation solution to run the remediation actions, which are dependent on the given infrastructure. Furthermore, results are visualized and alarms are triggered. Thus, administrators can intervene when they think it is needed. The focus in this thesis is the anomaly detection component within

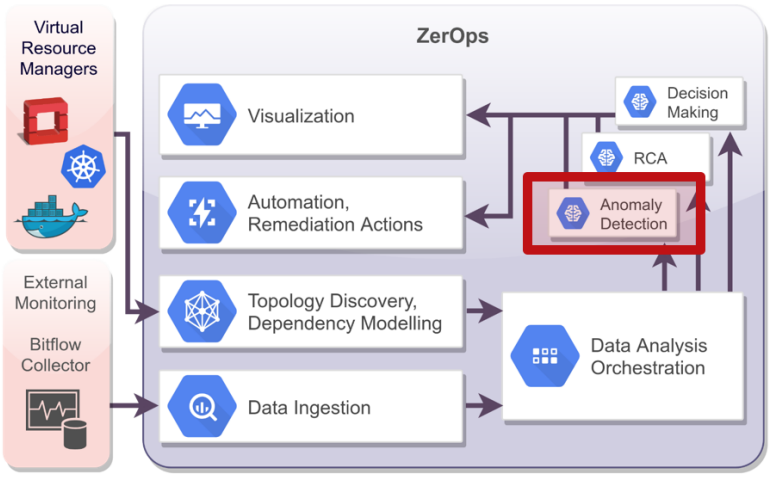


Figure 4.2: ZerOps architecture design including main capabilities, input and output components.

the self-healing pipeline, which is marked as red within Figure 4.2.

In summary, the focus of the thesis is a component-based AI solution for anomaly detection for triggering local events. Such events are processed in higher-level stages at the root cause analysis and remediation engine, which are out of scope of this thesis as we concentrate on the anomaly detection.

4.2 Categorization of AI-based Anomaly Detection

Providing an AI-based solution for anomaly detection, we need to differentiate between the learning phase and the prediction phase. While the learning phase is dependent on the type of machine learning model (ML model), the prediction phase is applied in the same online manner. Figure 4.3 illustrates the differentiation of both phases and the connections and dependencies among the generalized structure of applied ML models. From the data source, there is an incoming data stream of monitoring data, which in the prediction phase is sent to an existing ML model, providing a binary decision whether an anomaly was detected or not.

In the learning phase case, the ML model learning expects to provide hyperparameters to the ML model. Based on the historic monitoring data and hyperparameter settings an ML model is trained and provided to a model update function. The model updater manages the replacement of the prediction model with the adapted ML model from the learning phase. The model updater may rely on concept change detection of the current data stream, degree of change of the predictive model, and newly trained model or further evaluations, testing the accuracy of the predictive and learned models. Furthermore, the updater can just apply frequent replacement intervals. For online learning ML models, the model updates on every single incoming data point and is exchanged constantly.

But machine learning models (ML models) have to be trained at first. As introduced in Section 2.3.1, machine learning algorithms can be grouped based on the availability of training data:

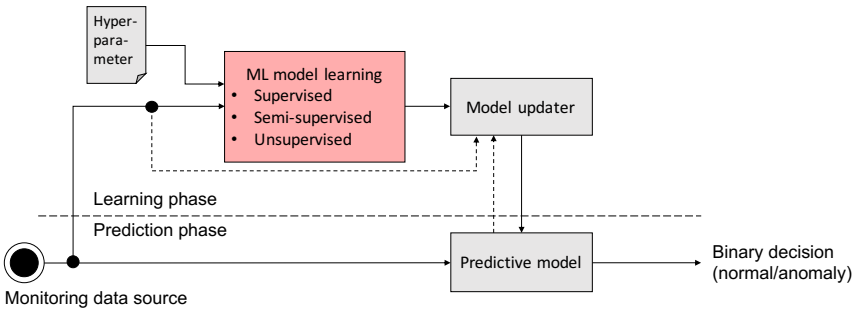


Figure 4.3: General structure for machine learning model embedding into productive system environments.

1. Supervised: There exist labeled information about all anomalies and normal behavior.
2. Semi-supervised: There exist labeled normal data for training, but knowledge of abnormal data is absent.
3. Unsupervised: There are no existing labels available, but assumptions are required about the structure of analyzed data.

Supervised Machine Learning Supervised behavior learning requires the existence of labeled data for both normal and abnormal behavior. There exist different supervised machine learning algorithms with respect to their aimed task like regression and classification. While regression focuses on predicting a value, classification aimed to provide classes as prediction results. For anomaly detection, we focus on binary classification. Thus, the classification result consists of the two classes normal and anomaly, while the input of the algorithm aspects the monitoring data as a multi-dimensional data point.

Figure 4.4 shows the general steps necessary to apply supervised machine learning techniques into production in an online manner. The learning phase of supervised models is separated and conducted as the first step. By gathering labeled data, a machine learning model is learned, validated, and tested before applying the trained model into production. As supervised models consume labeled training data, these are mostly not obtained automatically, but through human labeling. Therefore, such models should be very robust and consequently might not be updated in short intervals. For most machine learning algorithms, training models are computationally expensive. For this reason, the training data is expected to be collected together with labeled fault scenarios and normal system behavior in different load levels from the productive system or simulation environment. Given a trained supervised machine learning model, the model can be applied in production. The classification should be able to apply its computation just-in-time as those operations are mostly not computationally expensive. All in all, the training is conducted in an offline phase, while prediction happens in an online mode.

Semi-supervised machine learning Semi-supervised learning aims to model the normal behavior so that it can detect deviations from this behavior, which are consequently identified as anomalies. This type of learning is especially applicable for domains, where it is hard or even not possible to obtain abnormal data (e.g. nuclear power plant) or artificial injections would harm the system. Thus, one-class learning is applied to existing normal system data.

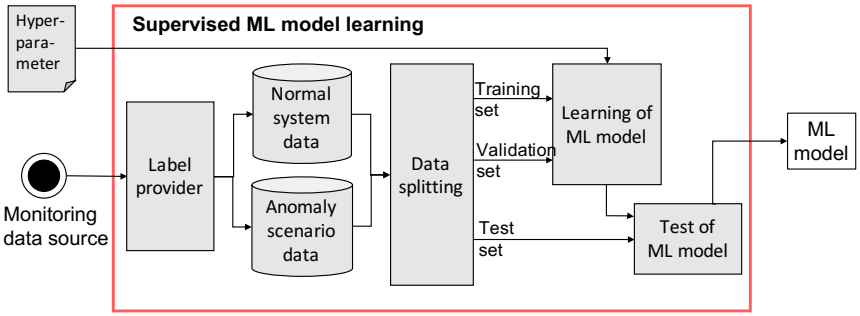


Figure 4.4: Learning phase of supervised machine learning models.

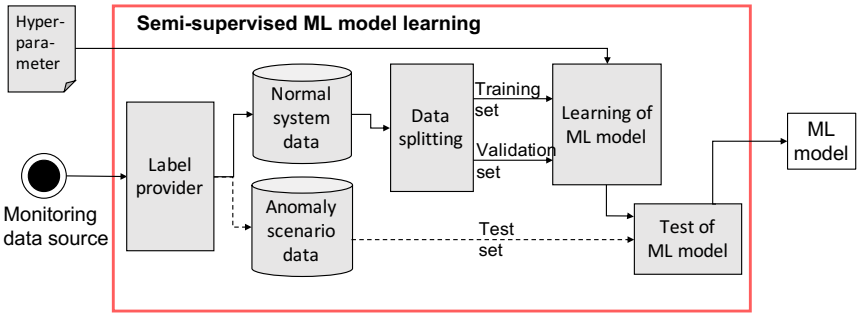


Figure 4.5: Learning phase of semi-supervised machine learning models.

As Figure 4.5 shows, normal data has to be labeled in order to make sure, that there does not exist any abnormal data for learning. In general, the semi-supervised model does not expect to know any abnormal system scenarios, but when a label provider is able to also create a database of anomaly scenario data, those can be used for ML model testing. Still this is optional, thus the learning of an ML model relies on a training and validation set of normal behaving data and provided hyperparameters. One-class learners train a decision boundary, which can be inferred by the data as only normal data is existing and sensitivity of the boundary is in general configurable by the hyperparameters.

Based on this training set of normally behaving data, strategies like clustering are among others utilized to capture this state. Due to the differentiation criterion, many well-known clustering techniques cannot be used as they map the complete space to clusters. For example, when learning a k-means clustering model (see Figure 4.6 left image) on normal data (black dots), the model cannot distinguish between normal and abnormal data points as for any data point a cluster can be assigned. This behavior is represented by the additional red dots, which shall be distinguished between normal and anomaly in Figure 4.6. In contrast to this, clustering models like BIRCH [52] (right image) provide additional spherical boundaries (dotted lines) around centroids. Thus, data points can be assigned to a cluster, indicating normal behavior, or not be assigned to any cluster, indicating an abnormal behavior. Thus, models can be chosen, which provide an estimation whether points belong to any cluster for binary decisions or probabilities whether including them in a given distribution.

Unsupervised Machine Learning Compared to the first two learning strategies, unsupervised learning does not rely on any labeled training data. ML models consequently need further assumptions (in general provided by hyperparameters) to

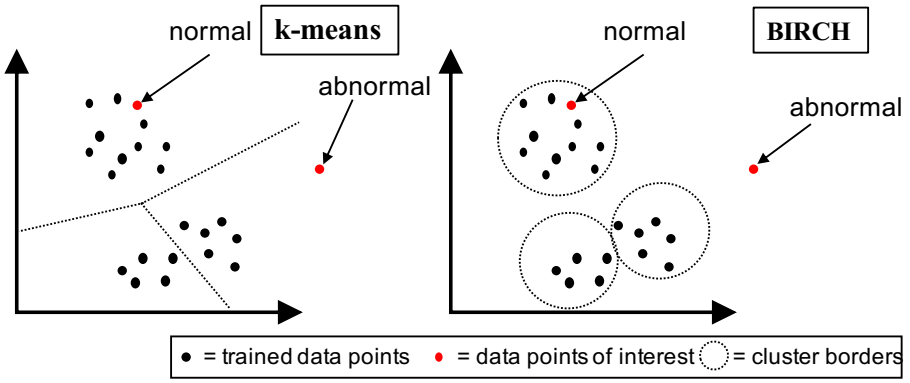


Figure 4.6: Example of three k-means clusters and clusters formed by applying BIRCH.

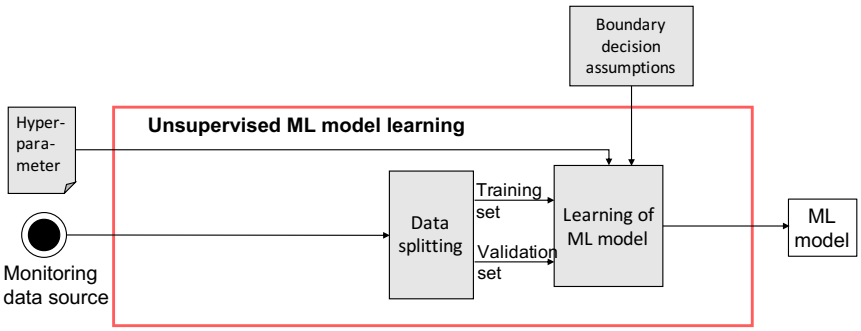


Figure 4.7: Learning phase of unsupervised machine learning models.

create accurate models. Assumptions can be for example either percentage of expected anomalies, percentage of the model, or signal changes. Still, the data can be split into training and validation set for either offline learning or in the case of online learning through seed learning with window-based storage of the latest data.

Recommendations for Applying Learning Strategies Supervised learning promises highest accuracy rates for anomaly detection as it has the advantage of labeled data. This technique is highly applicable when labels can be obtained automatically or costs of expert-based labeling is neglectable. Cost-effective labeling is the case when applied in a constrained set of software components where concept shifts rarely happen (normal behavior is known) and there exists a finite set of monitored anomalies.

In complex systems, knowledge bases of normal and abnormal labeled data do often not exist. Thus, semi-supervised and unsupervised learning strategies have to be applied. Semi-supervised learning enables the detection of unknown anomalies due to one-class learning the normal behavior. Still, labeling has to be performed as border definitions of the ML models are inferred from the labeled normal behavior. Likewise to supervised learning, semi-supervised learning should be advised to apply when normal behavior labeling is cost-efficient. This is again the case for automatic labeling strategies and when expert-based labeling is cost-efficient, which is the case for finite sets of software components without concept shifts.

Unsupervised learning does not require the existence of any labeled data. Consequently, unsupervised techniques can be applied at any times in order to differentiate

between normal and abnormal states. Likewise to semi-supervised learning, this type of learning should be advised to be applied in cases where anomalies are unknown, but concept shifts are also able to happen.

Providing an anomaly detection for black-box service monitoring for the ZerOps framework, we cannot rely on the knowledge of anomalies and have to consider concept shifts due to:

- service components may frequently change due to agile development,
- configuration of single service components and external service load varies in each service environment setup.

Consequently, the main focus is unsupervised techniques for the rest of the thesis. The main disadvantage still is the assumptions to be made in order to define a valuable border definition as well as the suffering of knowledge resulting in less accurate detection results.

4.3 Evaluation

Next, we show the applicability of supervised learning approaches assumed complete knowledge about anomaly situations and normal behavior patterns. Additionally, we present how semi-supervised learning suffers already from the lack of labeled anomaly information. Unsupervised learning is postponed to Chapter 7 for a larger evaluation.

4.3.1 Supervised Learning Evaluation

Evaluation Setup

The supervised machine learning algorithms J48, Random Forest, PART, REP TREE, LMT, JRIP, ONER, Random Tree, Hoeffding Tree, Decision Stump, SMO, and Naive Bayes (see Section 3.2) are included in the evaluation. We utilized the machine learning library WEKA [107], which consists of these algorithms from which we choose predefined hyperparameters suggested by the WEKA library.

Ten-fold cross-validation is performed on the cloud monitoring dataset as described in Section 7.1. The input for learning forms two datasets: the normal operation data and the anomaly data obtained through anomaly injection. Both datasets are shuffled and split into ten equally sized subsets. Randomly chosen pairs of normal operation subsets and anomaly subsets then form the ten test datasets. Thus, each training set consists of 28,610 data points.

Evaluation Results

Table 4.1 shows the results of this evaluation, sorted by accuracy. At the top, the approaches achieving the highest accuracy score are presented. *J*48 reached the highest score with more than 99%, followed by six algorithms approaching more than 98%. As we utilized the suggested hyperparametrization by the WEKA library, even more accurate results can be expected by applying hyperparameter tuning and feature engineering techniques.

Table 4.1 additionally illustrates the time for learning t_{learn} and predicting t_{pred}. The prediction time t_{pred} represents the time needed to apply the machine learning algorithm in prediction mode. The time shows the average time in milliseconds for an individual sample. The prediction results show the efficient computation as all algorithms are able to provide prediction in less than a millisecond. This is expected as the training phase is for most of the machine learning approaches computationally more expensive, while the prediction is mostly indifferent between the approaches and can be applied efficiently in an online manner.

Algorithm	t_{learn} **[ms]**	t_{pred} **[ms]**	**precision**	**recall**	F_1	**accuracy**
J48	3,030	0.0023	94.97%	99.81%	97.33%	99.05%
Random Forest	27,801	0.0984	93.21%	99.22%	96.12%	98.61%
PART	8,682	0.0039	93.41%	98.84%	96.05%	98.59%
REP TREE	1,943	0.0017	93.03%	99.11%	95.98%	98.56%
LMT	79,068	0.0039	93.16%	98.87%	95.93%	98.55%
JRIP	17,668	0.002	92.52%	99.15%	95.72%	98.46%
ONER	295	0.0015	92.37%	98.38%	95.28%	98.31%
Random Tree	755	0.0021	90.38%	97.91%	94.00%	97.83%
Hoeffding Tree	820	0.0036	87.29%	97.94%	92.31%	97.17%
Decision Stump	319	0.0018	86.26%	98.35%	91.91%	97.00%
SMO	18,796	0.0035	88.69%	75.03%	81.29%	94.02%
Naive Bayes	173	0.0129	92.83%	60.80%	73.48%	92.40%

Table 4.1: Quantitative and qualitative evaluation results for supervised learning.

Table 4.2 presents further details about the TP-rate, TN-rate, FP-rate, FN-rate, and AUC value as the dataset is imbalanced. The best results are achieved when the TP-rate and TN-rate reach 100%, while the FP-rate and FN-rate are consequently 0%. Given algorithms with above 95% accuracy also provide high rates as wished, also resulting in high AUC values. For this subset of algorithms, the AUC values are similar to the accuracy due to achieving such high accurate results. On the other hand, SMO and $NaiveBayes$ show high TN-rates indicating precise predictions for the normally behaving data, while the TP-rate is lower (75.03% and 60.80% respectively) compared to other classifiers (>97.91%) and does therefore not realize as accurate predictions for the abnormal data points. AUC values decrease therefore by a total of 7.5% (SMO) and 12.49% ($NaiveBayes$) compared to the accuracy.

The learning time t_{learn} represents the time (in milliseconds) needed to train a batch of 28,610 data points. In order to provide more guidance in selecting an appropriate machine learning algorithm, Figure 4.8 shows the dependency between learning time and accuracy. The x-axis represents the time, while the y-axis represents the accuracy. Based on these indicators, the preferred algorithm achieves zero training time, while reaching 100% accuracy. The upper left corner of the diagram represents such an ideal case. The figure shows, that there exist many algorithms in the left and upper part, marked as best choices surrounded by a zoom box. The zoom box (Fig. 4.9) illustrates further differentiation between the best options to chose a supervised approach.

Figure 4.9 provides the paretofront (the red line connecting circled ML approaches) recommending guidance on selecting the appropriate approach with respect to training time and accuracy. The $J48$ classifier provides the highest accuracy, while $ONER$, $REP\ TREE$, $Naive\ Bayes$ provide respectively decreasing accuracy, but also decreasing learning time (preferred). As described above, $NaiveBayes$ has the least capabilities to provide accurate predictions to the abnormal cases in contrast to the other algorithms in the paretofront. Thus, we recommend choosing one of the other three approaches when applying to similar use cases.

Algorithm	TP-rate	TN-rate	FP-rate	FN-rate	AUC
J48	99.81%	98.89%	1.11%	0.19%	99.35%
Random Forest	99.22%	98.49%	1.51%	0.78%	98.61%
PART	98.84%	98.54%	1.46%	1.16%	98.86%
REP TREE	99.11%	98.44%	1.56%	0.89%	98.78%
LMT	98.87%	98.48%	1.52%	1.13%	98.68%
JRIP	99.15%	98.32%	1.68%	0.85%	98.74%
ONER	98.38%	98.30%	1.70%	1.62%	98.34%
Random Tree	97.91%	97.82%	2.18%	2.09%	97.87%
Hoeffding Tree	97.94%	97.01%	2.99%	2.06%	97.48%
Decision Stump	98.35%	96.72%	3.28%	1.65%	97.54%
SMO	75.03%	98.00%	2.00%	24.97%	86.52%
Naive Bayes	60.80%	99.02%	0.98%	39.20%	79.91%

Table 4.2: Detailed evaluation results for supervised learning capturing the different rates (TP,TN,FP,FN,AUC).

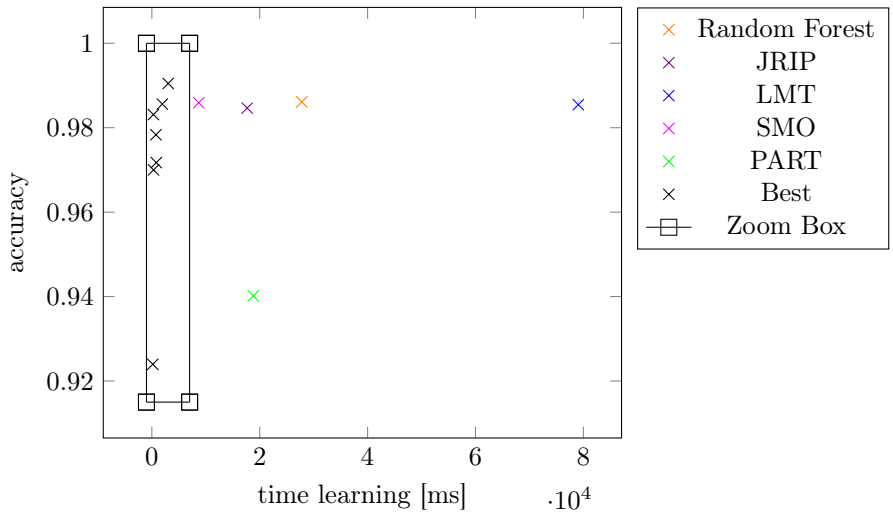


Figure 4.8: Performance of supervised approaches based on accuracy and runtime.

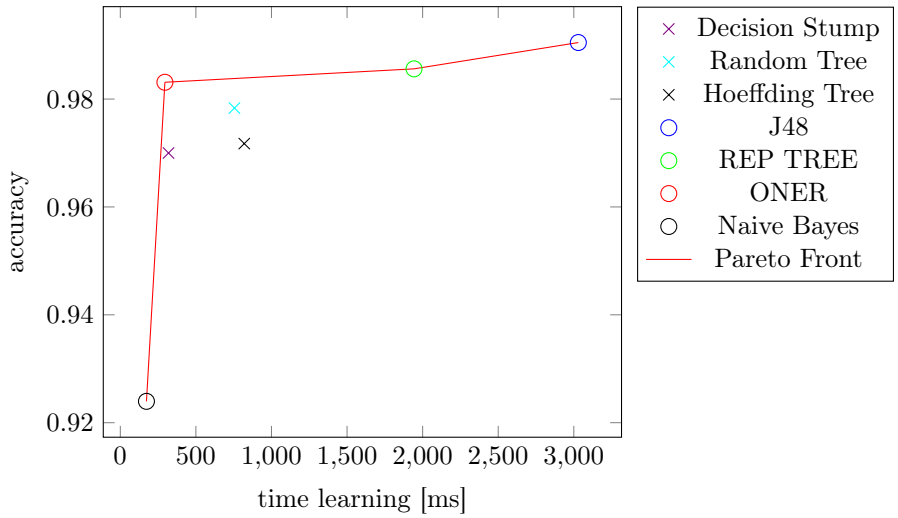


Figure 4.9: Zoom box showing the performance of supervised approaches in contrast of accuracy and runtime for selected algorithms in the upper left part of Figure 4.8.

4.3.2 Semi-supervised Evaluation

Evaluation Setup

We applied the following two clustering algorithms to the domain of anomaly detection for cloud service monitoring BIRCH [52] and BICO [47] (see Section 3.2 for details).

For implementation, we utilized JBIRCH by Roberto Perdisci[1] for BIRCH and the Java-based MOA (Massive Online Analysis) library[2] [260] for BICO.

The evaluation was performed on the cloud monitoring dataset as described in Section 7.1. Table 4.3 shows the applied hyperparameters for BIRCH and BICO.

BIRCH	initial threshold	0
	maximum number of nodes	10
BICO	initial threshold	0
	maximum number of nodes	10

Table 4.3: Parameter definitions for the semi-supervised clustering approaches BIRCH and BICO.

A random set of 1,200 data points (10min) is collected in order to meet the 1,200 data points on the basis of the complete knowledge of normally behaving data. For this, 100x the experiment was repeated to evaluate many different possible settings of choosing normal data points. The number of 1,200 data points was investigated through grid search (interval of [100,10000] data points through applying steps of 100) in order to find an appropriate number of data points, which does not cause the CF tree to grow too much for the given hyperparameters of the tree. Thus, we show next

[1] `https://github.com/perdisci/jbirch`
[2] `https://moa.cms.waikato.ac.nz/`

Algorithm	t_{learn} **[ms]**	t_{pred} **[ms]**	**precision**	**recall**	F_1	**accuracy**
BICO	2	0.1221	92.68%	64.02%	75.73%	91.65%
BIRCH	2	0.0074	48.25%	78.90%	59.88%	78.49%
J48	3030	0.0023	94.97%	99.81%	97.33%	99.05%

Table 4.4: Results for the first setup using randomized selection of training data showing runtimes and qualitative prediction results.

Algorithm	**TP-rate**	**TN-rate**	**FP-rate**	**FN-rate**	**AUC**
BICO	64.02%	98.71%	1.29%	35.98%	81.37%
BIRCH	78.90%	78.39%	21.61%	21.10%	78.65%
J48	99.81%	98.89%	1.11%	0.19%	99.35%

Table 4.5: Results for the first setup using randomized selection of training data showing the different rates (TP,TN,FP,FN,AUC).

just the results for the most accurate solution, which we were able to find through the grid search.

Evaluation Results

Table 4.4 shows the precision, recall, F_1 score, and accuracy for the BICO and BIRCH clustering algorithms applying the first setup with randomized training data input. The results show, that the precision, accuracy as well as the F_1 score is in those cases higher for the BICO algorithm, which is expected as BICO is an optimized version of BIRCH. Nevertheless, BIRCH shows higher recall indicating more precise predictions for abnormal data points. The supervised learning algorithm J48 was added to this table as comparison, outperforming both semi-supervised approaches. This is of course expected as the semi-supervised algorithms lack complete knowledge about any anomaly data.

The comparison for time to perform learning t_{learn} and prediction t_{pred} shows a clear differentiation between the semi-supervised and supervised approaches, although the presented three algorithms all internally apply a tree as data structure. While predicting t_{pred}, a single data point is the quickest for the J48. BIRCH just utilizes 0.0051ms more computation time. BICO shows the slowest computation with 0.1221ms, which is presumably quick enough when compared with a monitoring interval of 500ms. When comparing the time for learning t_{learn}, the semi-supervised algorithms outperform J48 as the incremental learning provides efficient computation.

Table 4.5 presents further details due to the imbalance of data. Again, J48 outperforms the semi-supervised approaches in all measurements. BICO shows higher TN-rates indicating more accurate representation of the normal behavior, while BIRCH is more precise in the detection anomalies.

Figure 4.10 illustrates the embedding of the supervised approaches compared with the first setup with the semi-supervised approaches. Like the supervised learning algorithms, the scatterplot aims to provide an overview of the most beneficial algorithms for applying those for the given domain. The scatterplot shows that BIRCH does not meet these requirements due to its lower accuracy, but BICO is placed within the zoom box.

Figure 4.11 shows in more detail the approaches highlighted within the zoom box.

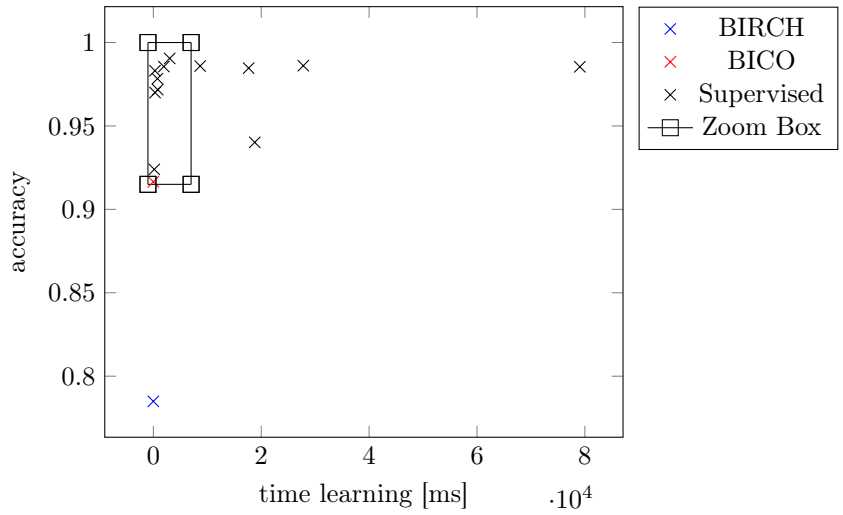


Figure 4.10: Comparison of the supervised and semi-supervised approaches with respect to learning time and accuracy.

Furthermore, the paretofront is marked and shows that BICO is a valuable candidate as it uses less learning time compared to any of the other approaches.

4.3.3 Summary

The supervised learning approaches provide the most accurate point-wise detection for the normal and abnormal cases and therefore show the applicability of such approaches. Thus, machine learning techniques are capable of learning accurate models even with further potential regarding hyperparameter optimization and feature engineering. Tree-based and rule-based classifiers (e.g. J48, Random Forest) show most accurate results, indicating that anomalies can be detected with deductive sets of complex rules. As supervised learning assumes the representative knowledge about the normal state as well as all different types of anomalies, these models should be applied for data streams, where such assumptions can be foreseen and met (e.g. for stateless functions with specificity defined usage patterns or for hard real-time services for embedded systems).

As the assumption of knowing all possibly causing anomalies is mostly difficult or not possible to ensure, semi-supervised approaches provide the possibility to model the normal behavior including border definitions to distinguish between normal and abnormal data points. Due to the absence of fault scenario catalogs and detection of unknown anomalies, semi-supervised approaches are applicable to many production scenarios. We showed, that when providing complete knowledge of the normal data to the learner, BICO is able to perform an accuracy of 91.65%, while providing much less learning time compared to the supervised algorithms. But still, there is a significant drop in differentiating normal and abnormal behaviors with respect to the accurate supervised approaches.

Both options are valid and applicable to the given problem, but the first option introduces further computations through an algorithm running parallel to the anomaly detection and therefore produces further overhead, which is not aimed with the goal of decentralized anomaly detection for the AIOps platform. In the following chapter, we develop an unsupervised anomaly detection approach based on the semi-supervised BIRCH.

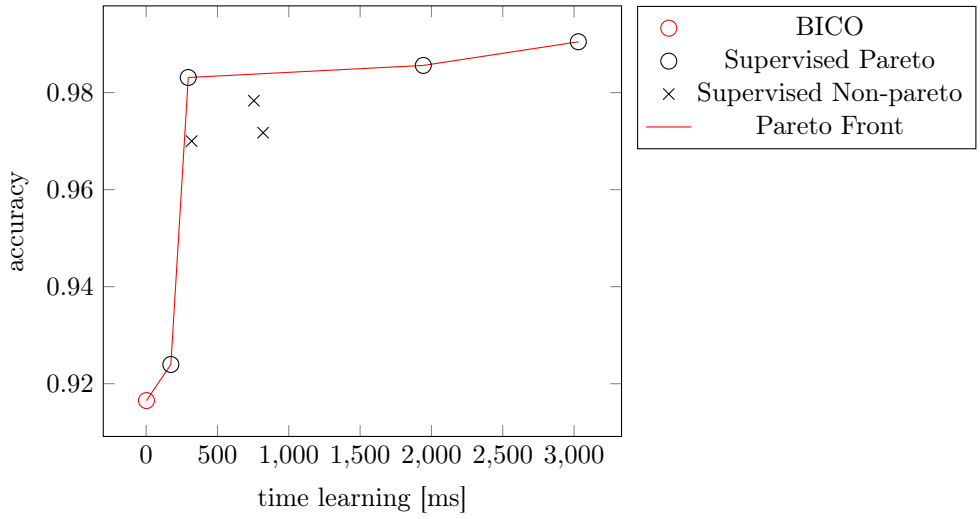


Figure 4.11: Zoom box for Figure 4.10 showing the paretofront.

Chapter 5

Concept Adapting BIRCH

We aim to provide a solution for unsupervised anomaly detection enabling zero-touch administration. This can be achieved through learning autonomously and continuously the normal behavior of a system in order to provide concept adaptability to the system domain.

5.1 Concept Adapting BIRCH

We introduce Concept Adapting BIRCH using the aging of clusters, making it possible to adapt to future concept changes. As we aim to provide unsupervised normal behavior learning on a possibly endless data stream of monitoring data, Concept Adapting BIRCH can be used as it covers the positive properties of both efficient iterative updating the internal model and adapting to the time series behavior.

We modified the order of the main four phases of BIRCH (see Section 2.3.2) and introduced an additional phase. Furthermore, we do not consider the 2^{nd} and 4^{th} phases as they are optional and use an offline approach. We consider the following three phases:

1. Inserting a data point: As described in [9], new data points are added to the CF-tree until the maximum number of nodes are reached, which are defined by the user. Reaching the maximum triggers the rebuilding phase of the tree. While rebuilding, the user-defined threshold is updated dynamically from then on. As defined by Zhang et al. [9], the updated threshold is approximated based on the size of the root CF entry using linear regression. The rebuilding phase is also called when the decay of micro-clusters results in removing CFs.
2. Predicting corresponding cluster: The original BIRCH algorithm first trains the clustering model with the complete dataset. After the trained model is prepared, each data point from the set is used to fit the model, while not changing it anymore. To use Concept Adapting BIRCH on a time series, we apply pointwise predictions. Thus, after training the current data point to adapt the CF-tree, we return the inserted cluster as a prediction.
3. Aging of micro-clusters: The new phase of decaying micro-clusters considers a continuous forgetting of clusters over time, denoted as aging. The aging allows us to forget older micro-clusters and strengthen the model adaptation to changing concepts. Aging can cause the deletion of nodes. Consequently, the rebuilding of the tree is applied to rebalance the tree structure. The aging procedure is described in more detail in the following.

5.1.1 Micro-cluster Aging

We introduce micro-cluster aging by applying two steps, consisting of a decay step followed by a removal step:

- Density decay: Let us assume to use the original BIRCH model to train a CF-tree on a data stream. By continuously inserting new data points into the CF-tree, the number of data points within CFs increases over time. Due to the constant growing tuple entries of clusters, older entries still have the same impact on the shape of the cluster. Thus, in the beginning, the data points have a greater impact on the shape of the CFs, when compared to the changes later in time. In order to keep the micro-clusters adaptable to concept changes, a decay function is applied to the micro-cluster giving new data points more impact on the cluster's shape.
- Removal of clusters: Older and now irrelevant data should not impact a model as much when considering concept adapting behavior over time. Therefore, BIRCH needs the ability to remove older clusters. After removing CFs, a rebuild of the CF-tree is performed to ensure the correctness of the micro-cluster entries for parent nodes.

The aging process should be applied to forget the history of past behaviors, but the model should be still fixed in memory and time. Storing historic data is not considered as this assumes a higher resource usage in memory. How this can be achieved is described next in detail.

Density Decay

Modifying the clustering method to provide concept adaptability to the change of data over time, we introduce a density-based decay function f_d. It defines the amount of changing a CF in its density. The change is applied on each leaf node, in order to give new incoming data points more influence on the shape of the cluster. The function f_d can be defined by the time, the number of points of the cluster or any other model or time-related metric, to let the model evolve with the data. We focus on time-related decays, such that at each time step, the number of points is decreased by ϵ and ϵ points are removed from the center $\frac{L}{N}$ of the micro-cluster, where $\epsilon := f_d$, as defined by Equation 5.1.

$$CF_t = CF_{t-1} - CF_\epsilon = (N_{t-1} - \epsilon, L_{t-1} - \epsilon \cdot \frac{L}{N}, S_{t-1} - \epsilon \cdot \frac{S}{N}) \tag{5.1}$$

Equation 5.1 provides concept adaptability. Thus, CABIRCH CFs remain the same in their structure (centroid and radius remain the same, but the change for the structure is larger for new incoming data points. Clusters are removed over time, which are not updated frequently anymore.

Through Equation 5.1, the structure of the micro-cluster does not change in the position of the centroid and the radius.

We show, that given the CF_{t-1} of time $t-1$, has the same centroid $c_{t-1} = \frac{L}{N}$ and radius $r_{t-1} = \sqrt{\frac{N \cdot \frac{L}{N}^2 + S - 2 \cdot \frac{L}{N}^2 \cdot L}{N}}$ as after applying the decay step. Let the centroid after the decay step be called c_t and the radius r_t . As defined by Equation 5.1, the centroid after the decay results in:

$$c_t = \frac{L - \epsilon \cdot \frac{L}{N}}{N - \epsilon} = \frac{L \cdot (1 - \frac{\epsilon}{N})}{N \cdot (1 - \frac{\epsilon}{N})} = \frac{L}{N} = c_{t-1} \tag{5.2}$$

For the radius:

$$
\begin{aligned}
r_t &= \sqrt{\frac{(N-\epsilon)\cdot\frac{(L-\epsilon\cdot\frac{L}{N})^2}{N-\epsilon} + (S-\epsilon\cdot\frac{S}{N}) - 2\cdot\frac{(L-\epsilon\cdot\frac{L}{N})^2}{N-\epsilon}\cdot(L-\epsilon\cdot\frac{L}{N})}{N-\epsilon}} \\
&= \sqrt{\frac{N(1-\frac{\epsilon}{N})\cdot\frac{L(1-\frac{\epsilon}{N})^2}{N(1-\frac{\epsilon}{N})} + S(1-\frac{\epsilon}{N}) - 2\cdot\frac{L(1-\frac{\epsilon}{N})^2}{N(1-\frac{\epsilon}{N})}\cdot L(1-\frac{\epsilon}{N})}{N\cdot(1-\frac{\epsilon}{N})}} \\
&= \sqrt{\frac{N(1-\frac{\epsilon}{N})\cdot\frac{L}{N}^2 + S(1-\frac{\epsilon}{N}) - 2\cdot\frac{L}{N}^2\cdot L(1-\frac{\epsilon}{N})}{N\cdot(1-\frac{\epsilon}{N})}} \\
&= \sqrt{\frac{N\cdot\frac{L}{N}^2 + S - 2\cdot\frac{L}{N}^2\cdot L}{N}} \\
&= r_{t-1}
\end{aligned}
\tag{5.3}
$$

Through Equation 5.1, the density of a micro-cluster is changed, and inserting new points into the micro-cluster results in a different structure after applying the decay function than not applying it, when not inserting a centroid as a data point into the micro-cluster.

We therefore show the change of the centroid with respect to applying decay against without using the technique. Let therefore, the centroid using decay be c_a while not applying the decay c_b. Let $c_a = c_b = \frac{L}{N}$. The newly added data point is described by x. Thus, $c_b = \frac{(L+x)}{(N+1)}$ and $c_b = \frac{(L+x-\epsilon\cdot\frac{L}{N})}{(N+1-\epsilon)}$ after also applying the decay, based on the additivity property of Equation 2.1. Let the difference of change be described by $c_b + \Delta = c_a$, where we show $\Delta \neq 0$, when $x \neq \frac{L}{N}$:

$$
\begin{aligned}
\Delta &= \frac{(L+x-\epsilon\cdot\frac{L}{N})}{(N+1-\epsilon)} - \frac{(L+x)}{(N+1)} \\
&= (L+x-\epsilon\cdot\frac{L}{N})\cdot(n+1) - (L+x)\cdot(N+1-\epsilon) \\
&= NL + Nx - \epsilon L + L + x - \epsilon\frac{L}{N} - NL - L + \epsilon L - Nx - x + \epsilon x \\
&= \epsilon x - \epsilon\frac{L}{N}.
\end{aligned}
\tag{5.4}
$$

If $x = \frac{L}{N}$ then $\Delta = 0$ and if $x \neq \frac{L}{N}$ then $\Delta \neq 0$. The change of radius can be shown in the same way and shows the same implications for $x \neq \frac{L}{N}$ and $\Delta \neq 0$. Thus, we showed that there is a difference in change of the structure when using Concept Adapting BIRCH's Equation 5.1 compared to applying just the BIRCH's inserts.

Equation 5.1 is valid due to the additivity property of the micro-cluster tuple as shown in Equation 2.1. It does not change the size and radius of the cluster but only its density. Removing points from the hull of the n-sphere, or from somewhere else other than the centroid, influence the shape of the cluster, which is not desired. This results in an incorrect structure of the CF-tree as clusters are no longer correctly placed within the CF-tree. By reducing the density of a micro-cluster, new data points, which are inserted into the cluster, have a higher influence on the shape of the cluster. Additionally, CFs that decay to a number of zero (or less) points in N are removed from the node. Consequently, leaf nodes without any CFs are removed from the CF-tree structure. Therefore, clusters require frequent insertion of new data points to maintain their existence.

Through Equation 5.1, clusters, which are not updated frequently anymore, are removed over time. The density decay is performed after inserting and predicting the current data point, so that after the end of every time step micro-clusters decay. When clusters have less or equal zero data points after decay inside their CF, those CFs are

removed, and the entire CF-tree is rebuilt. Assuming that a CF is not represented by the current data stream anymore, such a cluster can only remain available when the insertion of new incoming data points is larger than the decay of N. As we assume, that there are no represented points anymore for this cluster, the cluster will be removed when N is negative.

How constant updates of the micro-cluster influence the shape over time depends on the concrete definition of f_d. Therefore, we define the next different possible decay functions.

Decay Functions

Next, we define four types of decay functions, which differently impact the aging of micro-clusters over time:

- Decay based on thresholds
- Logarithmic decay
- Exponentially bounded decay
- Logistic decay

While the decay based on a threshold is configured through model parameters, the logarithmic, exponential, and logistic decay are dependent on the time. After defining the decay functions, we summarize and conclude the purpose for using these decay functions in the context of normal state representation.

Decay based on Thresholds A straightforward approach for defining a decay function is by specifying a single static value $K \geqslant 0$ as decay independent from time. Through constantly applying the decay, the cluster shrinks in its density. The definition of the value K is thereby crucial. For example, high values would lead to quick removal of clusters, while too low values do not impact the aging process at all. As such a value can be user-defined or based on initial model parameters, we introduce possible options to automatically chose K.

For the density function f_d, the probability of updating a single CF P_{update} can be applied. Thus, assuming a uniform distribution for updating a single CF (just for the leaf nodes) within the CF-tree, we can define $P_{update} = \frac{1}{L}$, where L denotes the number of leaves. As this number might change over time, we define L to be the maximum expected number, defined by the branching factor b and maximum height h of a given tree, which is defined by the user. Thus, $L = b^h$ and

$$f_d = P_{update} = \frac{1}{b^h}. \tag{5.5}$$

As f_d assumes uniform distribution, this form might not capture the realistic distribution of a given use case. Clusters with many inserts get concept change inflexible, while clusters with a small number of inserts disappear quickly. Therefore, we introduce time-dependent decay functions next.

Logarithmic Decay Decay through defining a value gives new emerging clusters just chances of being not removed when the update frequency at the beginning is higher than the average of a cluster. Otherwise, those clusters disappear quickly as initial values of N are always small at creation time. In order to capture this starting behavior of cluster variables, the decay function should capture such a behavior also. Let μ denotes the average number of inserts into every CF, b the tree width, β the growth rate and t the time after creating a cluster. A logarithmic function can be defined capturing the structure (as the height h of a balanced tree is defined by $h = \lceil log_b(L) \rceil + 1$) and time dependency of a tree:

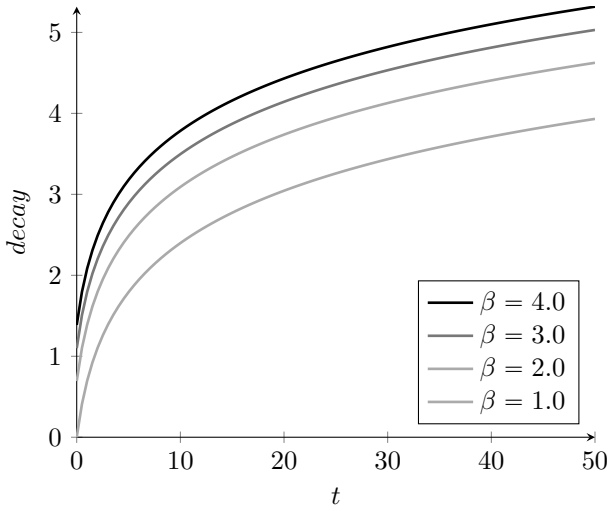


Figure 5.1: $\mu = 1$ and $b = e$

$$d_{log} = \mu \cdot \log_b(\beta \cdot (t + 1)) \tag{5.6}$$

As shown in Figure 5.1, different options for defining logarithmic decay functions are illustrated. At $t = 0$ all functions represent the initial behavior of an emerging cluster with low values in both N and r. Thus, the decay is also small. Over time, the decay increases constantly in logarithmic behavior.

The growth of the decay value is not bounded as the functions grow to infinity, so that over time, the decay increases higher than a cluster can grow. Thus, old clusters are always removed. In order to limit the growth, next we introduce the exponential decay.

Exponentially Bounded Decay As the name already indicates, the logarithmic decay function (see Equation 5.6) uses a logarithmic function as base function. The exponentially bounded decay is shown in Equation 5.7 and relays on an exponential function, which is often used to model economical [261] and biological growth [262] as the function can be bounded in its maximum rate. For example, bacterial growth shows that the replication of organisms is exponential, but due to bounded habitat, the death rate of organisms increases, so that both replication and death stabilize at a plateau as shown in Figure 5.2.

$$d_{exp} = \mu \cdot (1 - b^{-\beta \cdot t}) \tag{5.7}$$

Thus, we can define the maximal decay μ through Equation 5.5. Let b denotes the branching factor, β the growth rate and t the time. Figure 5.2 illustrates the behavior of an exponentially bounded function (Eq. 5.7) with different growth rates, but all bounded by a same maximum of $\mu = 1$. The behavior shows, that after a few time steps the maximum is already reached quickly as the function is based on a fixed basis b and a changing exponent t. Thus, in the field of biological growth, there exists a solution to provide a balanced increase in growth, which is also bounded by a defined limit, called the logistic function, which is introduced next.

Logistic Decay Logistic functions are widely used in different fields like bacterial growth cure predictions [263], Stock market analysis [264] and fish population modeling

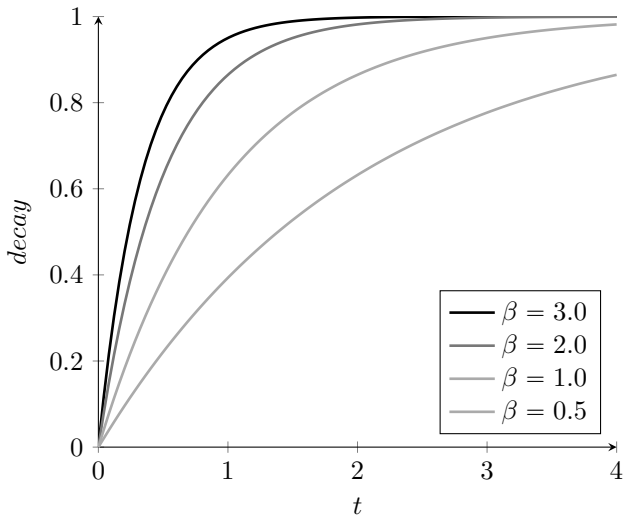


Figure 5.2: $\mu = 1, b = e$

[265]. In order to define the logistic function, we use the generalized logistic function [266] seen in Equation 5.8. Let K describe the maximal threshold for the decay, which is again defined by μ through Equation 5.5, while the lower asymptote is set to zero. Furthermore, let the parameters $C = 1$ and $v = 1$, which influences at which value the maximum slope is reached. Lastly, as Q defines what the value of $Y(0)$ is, we set $Q = (-C + \frac{(K-A)}{y_0})$ so we can directly define $y_0 = 1$ as shown in Equation 5.9.

$$Y(t) = A + \frac{K - A}{(C + Q \cdot e^{-\beta \cdot t})^{\frac{1}{v}}} \tag{5.8}$$

$$Y(0) = \frac{\mu}{1 + (-1 + \frac{\mu}{y_0}) \cdot e^{-\beta \cdot 0}} = \frac{\mu}{\frac{\mu}{y_0}} = y_0 \tag{5.9}$$

$$d_{logistic} = \frac{\mu}{1 + (\mu - 1) \cdot e^{-\beta \cdot t}} \tag{5.10}$$

This results in the function displayed in Equation 5.10, which decays micro-clusters to the average inserts μ, while also taking the time t aged after their creation into account. The rate of growth is dependent on the factor β as shown in Figure 5.3. In the beginning, new emerging clusters have the chance to get established as the decay value is small for low values of t. But later, when the decay increases due to the age of the cluster, the growth rate equals the maximum decay rate of μ.

Decay Functions Overview Table 5.1 presents a summary of the four defined decay functions in the two categories:

1. Impact of the decay on initial phase for emerging clusters
2. Impact of the decay on established clusters later in time

While the first threshold based decay functions cause quick removal of newly built clusters due to the static behavior of the decay compared to the growing characteristic of emerging clusters, the logarithmic, exponential and logistic decay functions provide the option to give emerging clusters the chance to get established over time and not be removed quickly after creation. In contrast to the logarithmic and logistic functions, the exponential decay increases quickly, thus emerging clusters have just little time to establish themselves. Of course, this behavior can be intentional, such that clusters

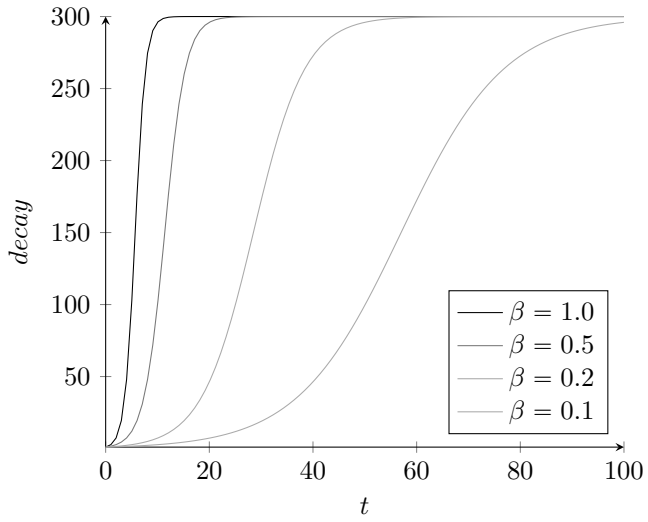


Figure 5.3: $\mu = 300$ and $b = e$

	Impact on initial phase for emerging clusters	**Impact on established clusters later in time**
Decay based on static thresholds	High impact (−)	Limited decay (+)
Logarithmic decay	Low impact (+)	Unlimited decay (−)
Exponentially bounded decay	Medium impact (o)	Limited decay (+)
Logistic decay	Low impact (+)	Limited decay (+)

Table 5.1: Summary of decay function properties.

always are removed quickly to stay highly adaptable over time. But as we like to model the normal behavior of a system over time, we also need to capture the cluster for a longer time frame. Thus, logarithmic and logistic decay show a moderate impact on emerging clusters.

Later in time, the behavior of the functions is different. While the threshold, logistic, and exponential decays are limited in growth, the logarithmic function does not bound the growth of the decay. Thus, older clusters will always be forced to be removed, even when the cluster is frequently updated. Again, this can be intentional to provide high adaptability for current context but is not favorable to be used for time series based anomaly detection when older clusters might still influence the normal behavior of a system.

All in all, the logistic decay provides the best options as the impact is low in the initial phase and, later in time, the impact is limited. Thus, emerging clusters can be established, while the rate of decay is bounded and it is not too drastic to automatically remove old clusters later. In contrast, clusters with frequent inserts are kept within the model.

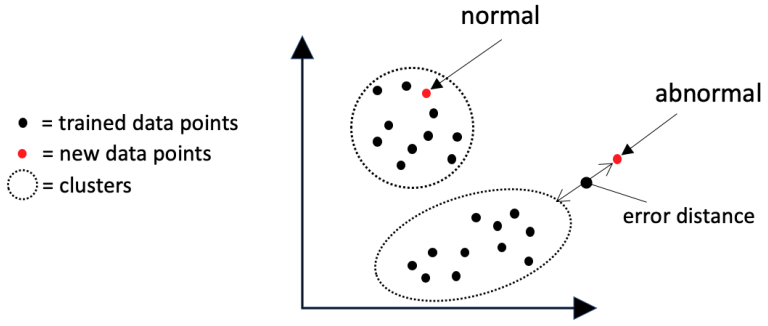


Figure 5.4: 2D example of trained clusters and the principle of anomaly detection.

5.2 Anomaly Detection using Concept Adapting BIRCH

Concept Adapting BIRCH is applicable for modeling the normal behavior of a system. This can be achieved when assuming most data points are normal, so that we can continuously train the model with the incoming data points. For anomaly detection, we swap the first two phases of Concept Adapting BIRCH to predict, whether the data point belongs to any cluster and then to insert the data point into the current CF-tree. If the data point is not contained within any cluster, the data point is considered to be an anomaly. Otherwise, the data point is considered to be normal. This enables to detect anomalies at their beginning, but due to the concept adaptation, longer running anomalies will be considered likely to be normal as they build new clusters within the CF-tree. The differentiation between normal and abnormal data points by the clustering result is illustrated in Figure 5.4.

As we try to remediate an anomaly after detection, we assume that anomalies disappear after a while. We also assume that continuously appearing anomalies, which are not remediated, can be considered as normal behavior. Such that after adapting the model to the new concept, it is possible to discover further anomalies within the signal.

Whether a data point is abnormal or normal, the approach can also quantify how large the difference to the normal behavior is when having a non-fitting data point. We define for this error distance an error-to-normal metric ϕ (Equation 5.11), which calculates the Euclidean distance from a given data point x to the nearest cluster border P_b.

$$\phi = ||P_b - x||^2 \tag{5.11}$$

The border point is defined by $P_b = \frac{r}{||x-c||^2} \cdot (x - c)$, where c is the centroid and r the radius of a given cluster and x any point outside the cluster. When data points are normal and therefore the predicted point is placed within a cluster, ϕ is set to zero. The error-to-normal metric ϕ can be used for further analysis steps, like anomaly evolution prediction or root cause analysis as it captures valuable information when specific dimensions have large prediction values and can indicate the concrete cause of an anomaly.

5.2.1 Identity Function Threshold Model

The distance ϕ can be used for further differentiating nearby cluster points to be considered as normal even though they do not fit the cluster directly. Modeling such sensitive borders, we introduced the generalized concept of Identity Function Threshold Models (IFTM), published by us in [267]. It enables the modeling of complex normal behaviors for unsupervised online anomaly detection techniques, which function with prediction errors like Equation 5.11.

IFTM consists of two main stages in order to detect anomalies for degraded state anomalies within a running system in a unsupervised manner:

1. The *Identity function (IF)* recommends a value representing the normality of the current monitoring data stream. In order to do so, the IF learns the reconstruction of the given data stream, without knowing future events. The IF should be capable to learn online and compute point-wise decisions for a given multivariate data stream.
2. The *Threshold model (TM)* is applied to the output of the IF, distinguishing between normal and abnormal behavior by computing a threshold value. The threshold computed by the TM is based on historic IF outputs and should be also learned online and compute point-wise decisions.

Let function $s : \mathbb{R}^n \to \mathbb{R}^n, n \in \mathbb{N}$ perfectly represent a given data stream, so that for a given data point $x \in \mathbb{R}^n$ the prediction is always $s(x) = x$. The goal of the identity function $\xi : \mathbb{R}^n \to \mathbb{R}^n$ is approximating s by not knowing future data points as the stream might be endless. Thus, let $g : \mathbb{R}^n \to \mathbb{R}^n$ represents the approximation error function. Thus, $g(\delta, x) = \delta(s(x), \xi(x))$, where $\delta : \mathbb{R}^n \to \mathbb{R}^n$ represent an error measurement and the function g aims to approach $\overrightarrow{0}$.

Given a data point $x \in \mathbb{R}^n$, the identity function forecasts a value $x'_t = \xi(x_{t-1})$, where $x' \in \mathbb{R}^n$. Based on the prediction, the reconstruction error Δ (also known as prediction error) is computed using δ, as described above, which can be any kind of error function (also called distance- or loss function). Different error functions like the Jaccard index, root mean squared error, or Euclidean distance can be applied depending on the considered identity function.

Assuming the IF reconstruction errors are performed, a valuable threshold T has to be selected in order to differentiate normal and abnormal errors. We assume that the identity function's reconstruction error is low for normal behaving monitoring data, while for abnormal data, which might not be known yet, the reconstruction error is high. Thus, the IF precisely represents the normal behavior of the incoming data stream through learning continuously as we assume that most of the available data is normal. We assume that the reconstruction error Δ is normally distributed, also assumed by [65–67]. Consequently, we define a threshold based on the mean $\mu(\Delta)$ and standard deviation $\sigma(\Delta)$ of the complete historic values of Δ. Based on this assumption, we define a threshold of $T = \mu(\Delta) + c \cdot \sigma(\Delta)$. The parameter c reflects a sensitivity value, as the false alarms can be approximated through T. Obtaining the mean and standard deviation iteratively over time is straightforward and computed in constant time.

Figure 5.5 illustrates the phases of applying an IFTM approach. Based on the currently monitored data from the given data stream, the IF ξ reconstructs the given time series. The reconstruction error is computed based on the prediction and monitored values. Combined with the threshold, a binary decision can be made whether the current data point is abnormal or not. The calculated reconstruction error is further used to optimize the IF as well as the threshold model. In both cases, the functions can adapt to the current context, thus concept shifts can be captured as well as chronic problems. This can be helpful, as concept shifts might happen frequently and the IF should be adapted accordingly. Furthermore, the threshold model can adapt its threshold, when high errors occur for a long period in time (in case of

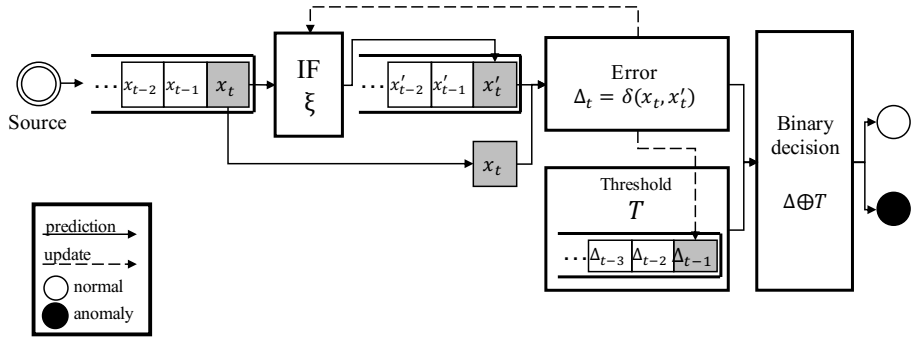


Figure 5.5: Input and output behavior of IFTM models.

chronic problems), where the anomaly impacts the system but does not cause a failure and remediations are not existing. For such cases, the anomaly detection is expected to adapt accordingly, but this is highly dependent on the monitored domain. In the case of IT-service monitoring, the anomaly detection should be aware of high frequent change of user behaviors, but should also accept chronic diseases when a lot of different remediation actions do not perform as wished. Through learning rate adaptions, both functions can be influenced to influence the rate of change. Thus, high rates can adapt very drastically to the current data and low rates do not influence the behavior at all anymore.

Due to the concept of updating the model based on a single data point, the method can be applied as online learning. How the IFs update their internal models is highly dependent on the machine learning model used and investigated in several of our papers [267–269].

Embedding CABIRCH into IFTM

Incoming data points are predicted by CABIRCH to return the error-to-normal metric ϕ, which is considered as a reconstruction error. In order to use the defined error function, CABIRCH does not need to return ϕ directly, but the border point for the nearest cluster, such when applying the Euclidean distance as error function would result in returning ϕ directly, but fitting the IFTM model. As a threshold function, one could then use a fixed threshold to be zero. This would imply the same approach of testing whether a data point fits into any cluster ($\phi \leqslant 0$) or not ($\phi > 0$), respectively whether a point is abnormal or not.

Equation 5.11 provides a metric, describing the Euclidean distance to the normal behavior of a component of interest. In order to provide more precise information of single dimensions, Equation 5.11 can be also applied to the single dimension values within the vector, resulting in distances for individual metrics. Both, the overall distance and the metric specific distances can then be forwarded to the analysis steps like RCA and the remediation component. These analysis steps can use this as additional information as these distances provide data about distances to expected normal behaviors, but are out of scope for this thesis and referred to as future work.

5.3 Evaluation

The evaluation aims to show at first the impact and suggested configuration options for CABIRCH and provides results for anomaly detection on the basis of the cloud monitoring dataset as described in Section 7.1.

5.3.1 Influence of Decay Rate Selection

Selecting a valuable growth rate parameter β is crucial for providing an accurate model of normal behavior. It highly depends on the reflected normal behavior. Consequently, a sequence of 20min of initial normal load from each monitored component is evaluated on which CABIRCH is continuously trained.

Table 5.2 presents the hyperparameter configuration for CABIRCH. The decay rate β is investigated in the range between 0.0 and 0.08 corresponding to 0 not applied decay (therefore same as BIRCH).

CABIRCH	initial threshold	0
	maximum number of nodes	20
	logistic function decay - max decay	1
	logistic function decay - β	$[0, 0.08]$, step size: $1 \cdot 10^{-3}$

Table 5.2: Parameter configurations for CABIRCH decay rate evaluations.

Figure 5.6 illustrates different β values compared to the average size of micro-clusters within a CF-tree and false alarm rate. The false alarm rate reflects the percentage of data points, which do not fit any cluster compared to all given data points. BIRCH ($\beta = 0$) achieves a micro-cluster size of 0.43, while the false alarm rate is 0.8%. Increasing β values cause the constant shrinking of clusters, while the false alarm rate rises. This is expected as the chance of smaller coverage of space causes a higher chance that the following data points do not fit any cluster due to a smaller radius. At $\beta = 0.077$, the false alarm rate rises to 100% due to the cluster size of close to zero. At this point, the decay value causes the forgetting of micro-clusters at such a high rate, that the CF-tree is unstable and clusters are not able to build up. Any further increase of β results in the same results from then on. At which point this change in stability happens is dependent on the maximal decay value set and the fluctuation of the captured signal. For example, when increasing the maximal decay value in this experiment, the β value has to be decreased in order to provide the same curve characteristics as the decay would otherwise be too drastic. At $\beta = 0.076$, the smallest cluster sizes (0.11) are achieved before the instability happens. In contrast, the false alarm rate rose until that point to a value of 25.4%.

The relationship between the cluster sizes and false alarm rate is presented in Figure 5.7. The illustration reflects a nonlinear behavior, but rather an exponential increase of false alarms with decreasing cluster sizes. By decreasing the cluster sizes by more than half (from 0.43 to 0.21), the false alarm rate increases by less than 3.6%.

In order to provide a further illustration of the impact of clustering result, we present in Figures 5.8 and Figure 5.9 a time-based side-by-side comparison between BIRCH and CABIRCH applied on a weather observation dataset from the CityPulse EU FP7 Project [270,271] representing concept shifting levels of carbon monoxide and ozone through the year. Overall, these three Figures show the expected behavior of providing smaller sized clusters, which are adapted to the data stream (having nearer distances to clusters). As clusters of BIRCH constantly grow until they cover large areas of space, the false alarm rate decreases over time. As the clusters for BIRCH remain on similar positions later in time, they never can cover the complete space as the n-spheres are not allowed to overlap. Thus, spaces between n-spheres exist and cannot cover new drifted data points due to inflexibility of adaption later in time. This behavior is shown through selected representations of BIRCH and CABIRCH over time from Figure 5.8 to Figure 5.9. Especially Figure 5.9 in subfigure (c), while subfigure (d) covers better the current data distribution with smaller clusters.

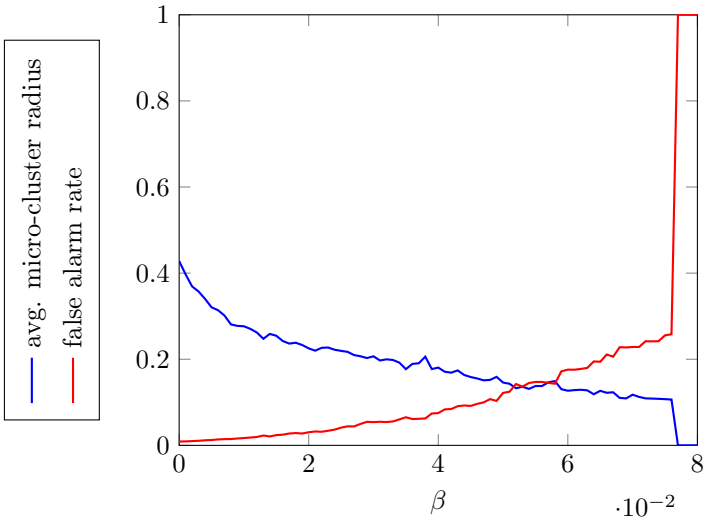


Figure 5.6: CABIRCH applied on cloud monitoring data under normal load situations and different β values.

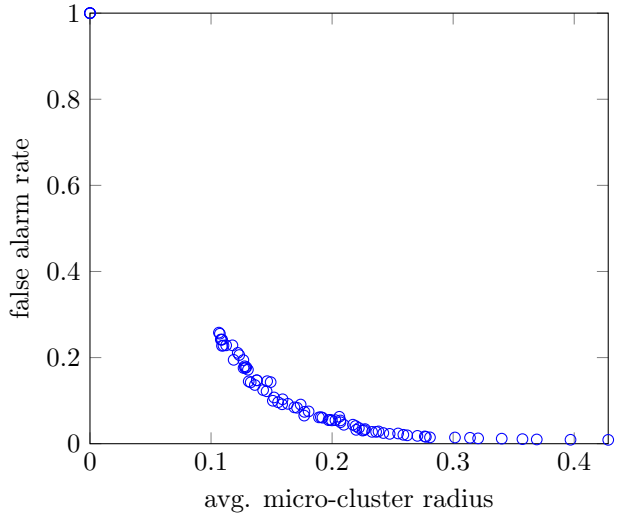
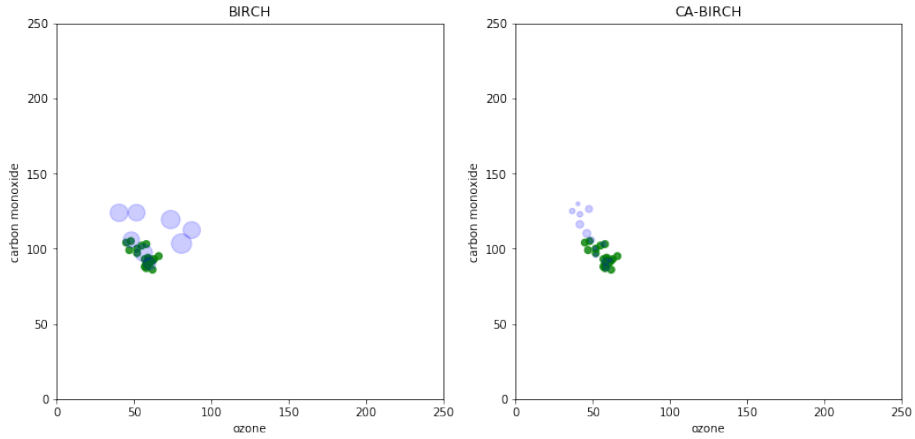
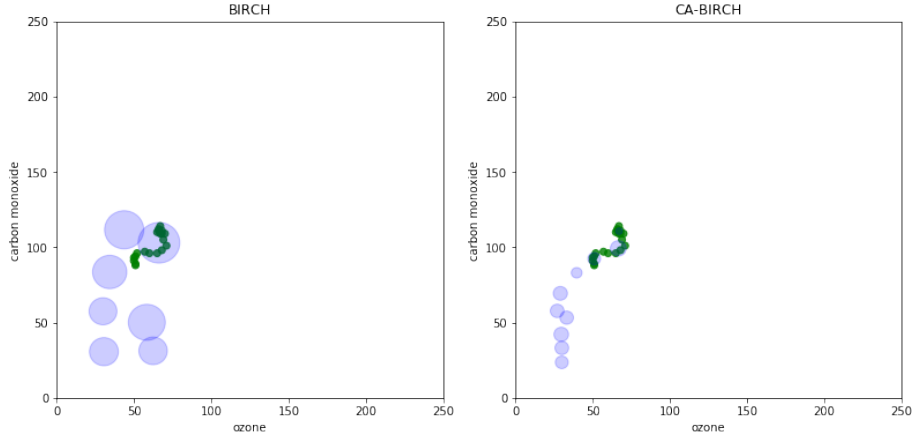


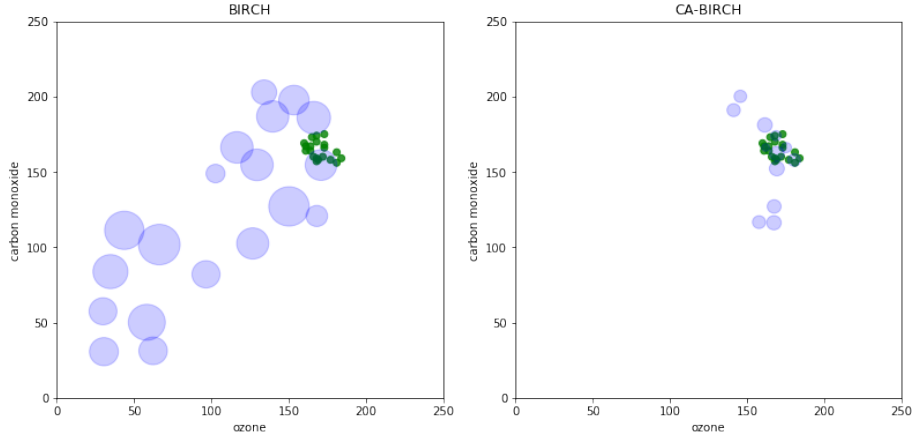
Figure 5.7: Comparison between false alarm rate and average micro-cluster size.



(a) BIRCH (b) CABIRCH

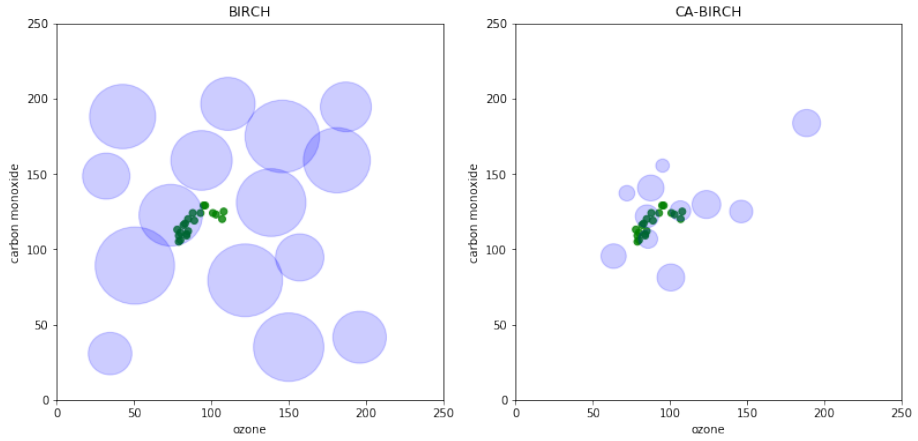


(c) BIRCH (d) CABIRCH

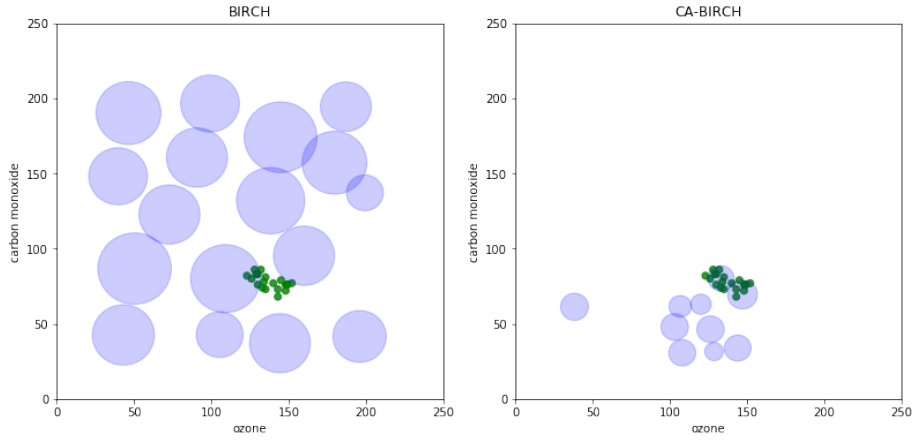


(e) BIRCH (f) CABIRCH

Figure 5.8: Illustration of BIRCH and CABIRCH micro-clusters (blue) and latest 20 data points (green) over time for the dimensions carbon monoxide and ozone of the pollution dataset. (a) and (b) at time $t = 100$. (c) and (d) for time $t = 1,000$. (e) and (f) for time $t = 2,000$.



(a) BIRCH (b) CABIRCH



(c) BIRCH (d) CABIRCH

Figure 5.9: Illustration of BIRCH and CABIRCH micro-clusters (blue) and latest 20 data points (green) over time for the dimensions carbon monoxide and ozone of the pollution dataset. (a) and (b) at time $t = 10,000$. (c) and (d) for time $t = 100,000$.

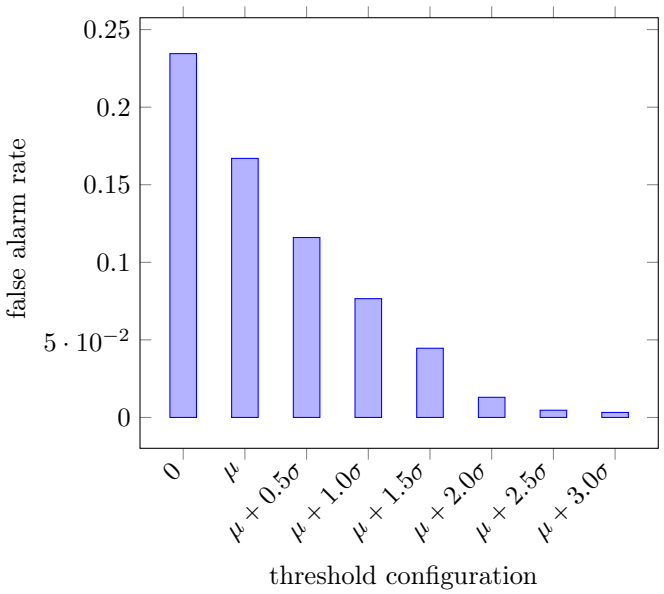


Figure 5.10: Relationship between the false alarm rate and configuration of the threshold model.

5.3.2 CABIRCH-based Anomaly Detection

The false alarm rate is decreased by applying the proposed IFTM procedure due to the dynamic threshold model. The threshold model provides a captured boundary between the border of the n-spherical micro-clusters and the threshold to be considered as normal. Figure 5.10 presents the impact of applying Gaussian distribution as threshold model with different parameters for the factor of σ when configuring CABIRCH with $\beta = 0.07$ and the rest of hyperparameters described in Table 5.2.

The first bar shows the relationship of the standard configuration of CABIRCH without making use of the dynamic threshold. This configuration realizes a false alarm rate of 23.5%. The following bars present a constant decrease of false alarm rate, as the threshold model constantly more incorporates the fluctuation of the prediction error of misclassified instances. The second bar μ does not incorporate the fluctuations, but the running average of prediction errors. The false alarm rate drops therefore by a total of 6.8%. Applying a factor of 2.0 to σ forces the false alarm rate to drop below 1%. The selected factor for σ reflects the sensitivity as it also increases the boundary to avoid potential anomalies to be recognized. Configuration with respect to abnormal data and normal data is of course recommended, but out of scope for unsupervised applicability. Thus, we consider a factor of 2.0 for the following experiments.

Figures 5.11 - 5.14 present the prediction error behavior when applying IFTM-based CABIRCH to a continuously simulated data stream. At first, the normal signal is simulated for 10,000 data points, followed by a presented (as blue signal) anomaly pattern consisting of 200 data points as presented in the x-axis. The anomaly patterns follow the characteristics of real-world anomaly patterns for cloud services as described in Section 3.1. Error A (red signal) represents point-wise continuous learning of CABIRCH, while error B (gray signal) shows the prediction error when applying the latest prediction model on basis of the first 10,000 data points. Anomalies are expected to be detected when the errors reflect increases in their values.

Figure 5.11 presents on the upper left diagram a rapid jump of the signal. The

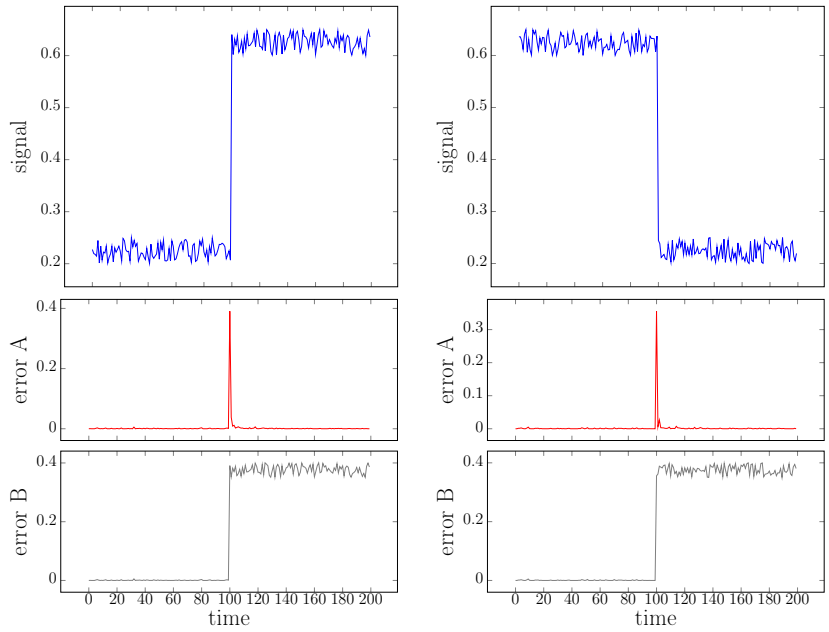


Figure 5.11: Prediction error behavior of CABIRCH for rapid changes with static increased or decreased anomaly pattern.

corresponding error A and error B are presented below, showing that error A detects the point of change, while error B captures the complete series after the anomaly pattern has started. The same behavior of errors is shown and expected for rapid drops as presented on the right side. Error A is capable of change point detection and adapts to the anomaly state quickly. Triggering those change points with high errors to remediate the anomaly relies on the always correct detection of such events. Error B captures the complete sequence of the anomaly state but does not adapt to the signal when this captures a chronic, normal state.

Leakage anomalies are represented by Figure 5.12. The signal successively increases (left) or decreases (right) over time. As the signal continuously changes, Error A shows higher error values for the phase of increase and decrease, while adapting again quickly when the signal follows a static behavior. Aggregating high spikes in a sliding window would ensure the chance to capture those phases of high fluctuating errors. In case of error B, the error values increase in intensity as the monitored signal moves away from the trained normal scenario. Error B represents therefore also a degraded phase capturing the degree of anomaly intensity, which might be beneficial for the remediation engine to select appropriate actions based on the severity.

In the case of fluctuation change within the signal (see Figure 5.13), we consider an increase of fluctuation around a base signal (left) and correspondingly a decrease of fluctuation (right). Error A reflects the change point when the fluctuation pattern changes with increased error intensities. After the pattern of the signal has changed, error A adapts to the signal and the error value decreases. Likewise to the rapid change of the anomaly signal, error A can be used for change point detection in fluctuation changes. Error B shows again a drastically increase in the error intensity when the anomaly pattern starts to change.

Figure 5.14 presents a sinus signal including two anomaly patterns within the signal in order to describe context-based anomalies within a seasonal behavior. Error A reflects both change points, when the anomaly starts and ends, while error B pro-

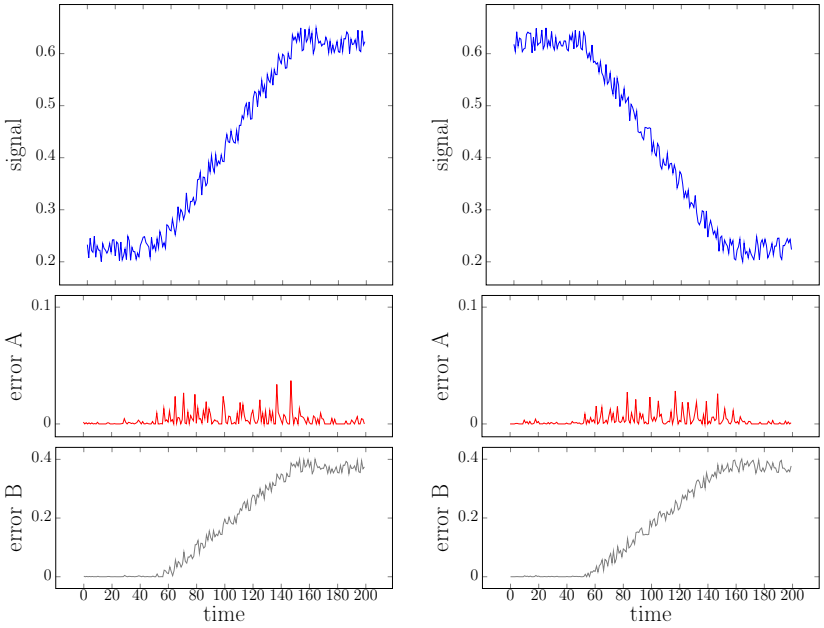


Figure 5.12: Prediction error behavior of CABIRCH for successive increasing and decreasing anomaly pattern.

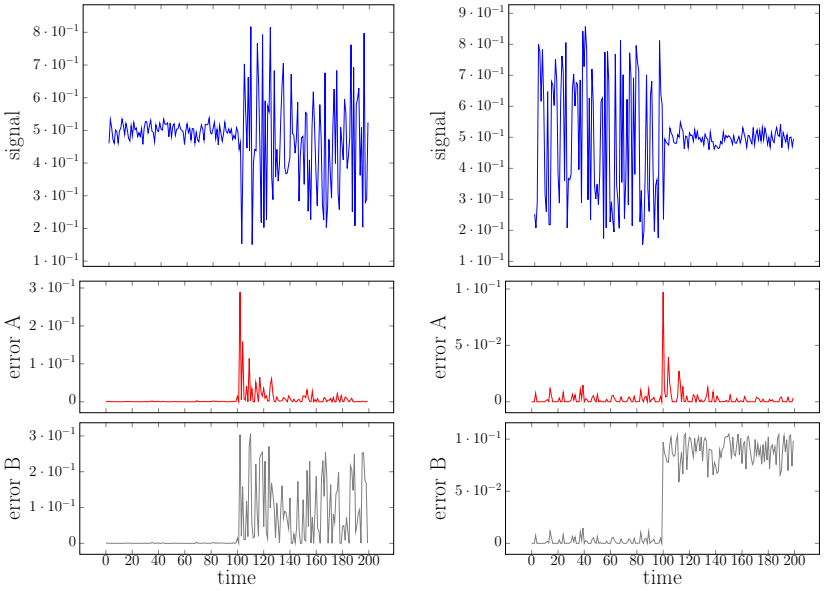


Figure 5.13: Prediction error behavior of CABIRCH for changing fluctuations as anomaly pattern.

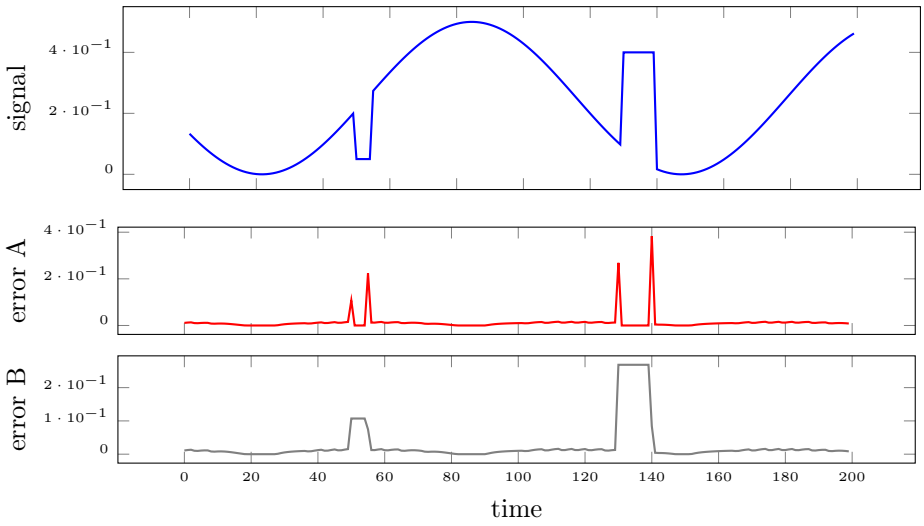


Figure 5.14: Prediction error behavior of CABIRCH for anomaly patterns within a sinus wave.

Algorithm	TP-rate	TN-rate	FP-rate	FN-rate	AUC
BIRCH-10fold	78.90%	78.39%	21.61%	21.10%	78.65%
CABIRCH	54.27%	96.69%	3.31%	45.73%	75.48%

Table 5.3: Results of CABIRCH compared with semi-supervised learning results of BIRCH.

vides high error values for the complete time frame of the anomaly. Furthermore, the intensity of error A and error B provide insights in the order of anomaly intensity.

Based on these findings, the anomaly detection is considered to be carried out in a tumbling window approach whenever it is aimed to capture the whole sequence of an anomaly. Anomaly detection is applied using a pretrained model from the previous tumbling window, while continuous training is performed for the next window. In cases, where anomaly detection is expected to provide change points, we recommend applying CABIRCH without tumbling window adaption.

Anomaly Detection Applied at Monitored Cloud Environment

We applied grid search to CABIRCH in order to determine optimal results with respect to the AUC value for anomaly detection on the cloud monitoring dataset (see Section 7.1). The search space for CABIRCH is described in Appendix B. We applied the step size $1 \cdot 10^{-3}$ for the decay function and $1 \cdot 10^{-2}$ for the maximum decay value and $[100, 200, ..., 1000]$ for the tumbling window size.

Table 5.3 shows, that CABIRCH reaches more than 96% in the TN-rate. Respectively, this describes a precise cover of the normal behavior. On the other hand, the TP-rate is small (54.27%) and therefore predicts roughly more than half of all abnormal data points correctly. Compared to results of BIRCH-10fold in the semi-supervised evaluation, the results show a drop of 3.17% in AUC value, which is expected due to the absence of complete knowledge of the normal behavior (CABIRCH is executed and trained online-wise).

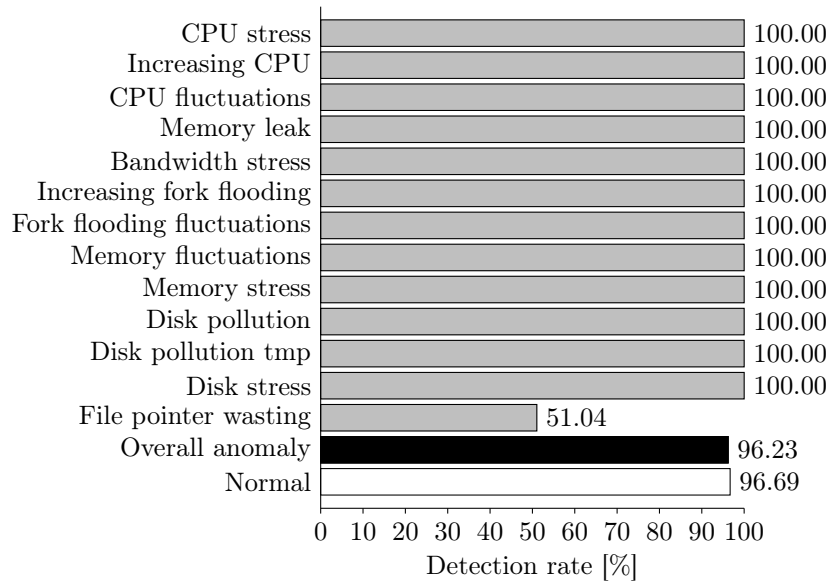


Figure 5.15: Event-based detection rates for the individual anomalies and normal data points.

As point-wise predictions are a hard indicator of accurate results, this evaluation metric does not represent the needs of an anomaly detection algorithm for self-healing purposes in production environments. For this, we investigate in more detail the event-based evaluation metrics. Due to the behavior rapid adaption of CABIRCH to the signal, tumbling window sizes have to perfectly represent the length of the abnormal sequences in order to reach high point-wise results, which is difficult in production. For each anomaly event, metrics are collected to show the rates of detecting anomaly events and times are compared when the anomaly was detected compared to the start of the anomaly.

Figure 5.15 presents the percentages of detected events for the different types of anomalies (gray) and the overall anomaly detection rate (black) for CABIRCH. Except the anomaly type file pointer wasting, all events were successfully detected, resulting in a 96.23% TP-events rate. In white, the normal detection rate is presented, which is point-wise and indicates a false alarm rate of 3.31% within the normal behaving phases.

In addition, the time difference from the start of the anomaly until it is actually detected by CABIRCH is represented in Figure 5.16. The average length of a false alarm is represented in white, which is less than 4 seconds. Again, the different anomaly types are represented in gray and are sorted based on Figure 5.15. Figure 5.16 shows that there are high differences when an anomaly is detected by the given algorithm. Successive increasing anomalies show larger detection times than rapid changing or fluctuating anomalies. This behavior can be explained by the adaption by CABIRCH.

Furthermore, Figure 5.17 shows the dependency between the average detection time for the different anomaly types, compared to the standard deviation from those. The plot shows a correlation of 0.893 (Pearson's correlation coefficient), which indicates a high linear dependency between the detection time and standard deviation. This means there does not exist a high difference in detection time for short average detection times, while for a larger average detection time, the fluctuation increases.

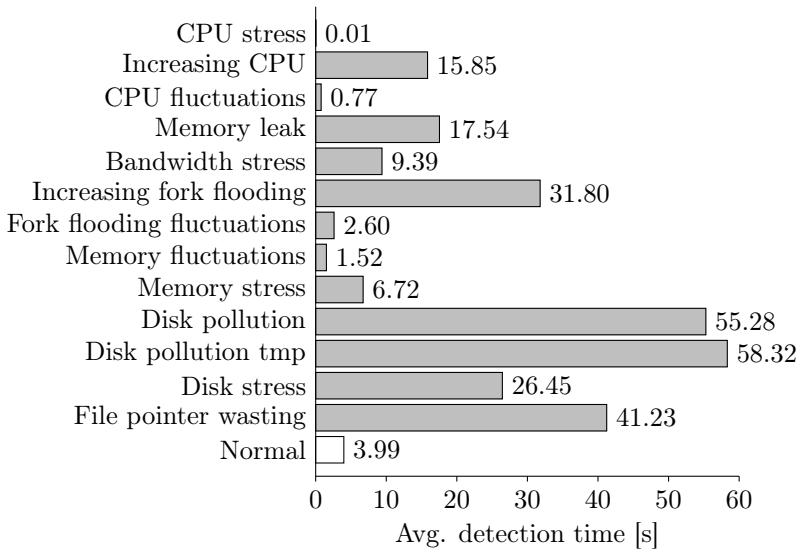


Figure 5.16: Event-based average detection times for the individual anomalies.

Based on these findings that such differences in detection time are described for successive increasing anomalies, the results indicate that the intensity of such anomaly types is relevant. Thus, non-intensive successive changes cannot be recognized.

The results show the applicability using CABIRCH for event-based anomaly detection with 96.23% TP-event rate and 3.31% false alarms. While 96.23% of anomaly events are going to be detected, point-wise TP-rate and detection times of anomaly types show the limitations of this approach. Successive changing anomalies are detected late, but accurate causing a small TP-rate. In the worst case, this might result in less options for risk-aware remediation action selection for productive environments due to late detection times. For the anomaly type of file pointer wasting, there exists further room for improvement. The next chapter continues to automatize for applying AI-based models by investigating the cold start problem of providing a valuable machine learning model from the beginning. For zero-touch administration, automatized optimization for selecting optimal hyperparameters is crucial as administrators are usually not AI experts, which is investigated in detail in the following chapter.

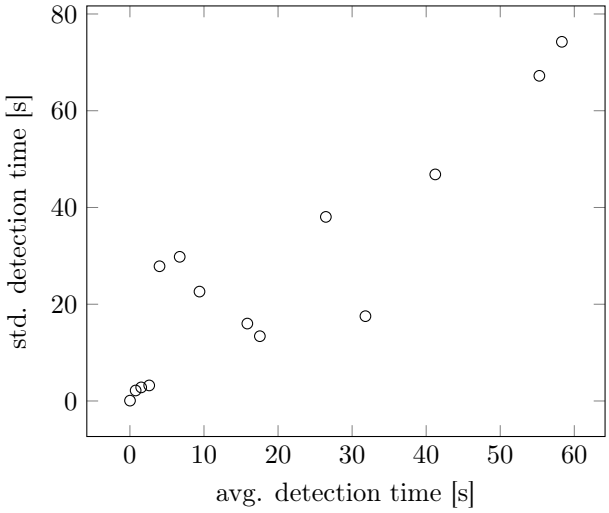


Figure 5.17: Comparison of average detection times and standard deviation of detection times.

Chapter 6

Cold Start-Aware Identity Function Threshold Models

This chapter focuses on the phase of initializing an IFTM-based anomaly detection model. The initial phase is referred to as cold start problem capturing the hyperparameter configuration and initial learning for machine learning models. As determining the best hyperparameters is crucial for achieving high accuracy for anomaly detection, it is difficult to find those beforehand in a productive system. This is due to the problem of collecting valuable data, which are not outdated and can be used for building robust models in dynamic environments.

In general, there exist two key aspects to approach to mitigate the cold start problem:

1. Selecting an already pretrained model (or generalized parts of pretrained models) from e.g. similar already deployed service, which is widely investigated in the context of robust models and transfer learning.
2. Optimizing the hyperparameters, if there does not exist a valuable pretrained model in the first place (see Section 6.2).

As shown in Algorithm 2, a suggested approach first checks the existence of a stored model for its particular component. In case of existence, this model is applied or otherwise continued with checking whether a similar model is available for usage. In case that there is no existing model of the component nor a similar model available, we propose a hyperparameter optimization methodology.

Algorithm 2 Cold start-aware approach for IFTM models

Input: *component*: component to be initiated
Output: *model*: IFTM model for anomaly detection
1: **if** Trained model exists for starting *component* **then**
2: **return** *model* ← load existing model
3: **else if** Trained model exists for similar component **then**
4: *models* ← load similar models
5: **return** *model* ← select best similar model out of loaded models
6: **else**
7: **return** *model* ← initiate model through hyperparameter optimzation

The definition of the term *similar component* refers to the consideration of models, which are either robust and therefore independent of any monitored component or are

applicable to transfer learning [272–274]. With the absence of any model, we still have to consider training an ML model from scratch, where the ML model has to be configured.

Hyperparameter configuration changes the functionality of the given machine learning algorithm. Determining beneficial hyperparameters for the aimed problem domain is crucial as it influences the method's quality of results. Even when applying the algorithm to a new component (this might be the change to another monitored service component), such hyperparameters must often be re-tuned [275]. Non-experts in machine learning require off-the-shelf solutions to configure hyperparameters as the degrees of freedom of choosing such are too high. Therefore, automatic approaches are required to evaluate and configure the machine learning algorithms, without the need for detailed knowledge about the exact machine learning algorithms.

The main problem for hyperparameter tuning is the complexity of the size of the search space. Let p_1, p_2, p_3 be hyperparameters for which the four values $\{a, b, c, d\}$ can be chosen. Consequently, for this setup there exist $3^4 = 27$ possibilities. In general, there is an exponential coherence p^n for a given number of parameters p and number of possible values n, assuming that all parameters have the same number of possible values. It gets even more problematic if there is not a finite number of values to choose from for a parameter, but e.g. an interval containing an infinite number of possible values, like choosing a real number between 0 and 1. Often interval ranges and step widths are defined in order to discretize such parameters and allowing to operate with a wide range of search approaches, like grid search, but in general this is not mandatory. Bergstra and Bengio [201] show that random search over non discretized features is more variable as through discretization the global optimum might not be reachable.

Due to the exponential complexity of enumerating all different possible parameters, there exist different methods beyond exhaustive search to determine the best parameters. For example, grid search is a form of exhaustive search, where the possible values are discretized and the range of values is finite. As grid search also performs a complete search on this set of defined discretized values, the optimum with respect to the given values can be found. Thus, the user has to shrink the set of values very much to provide a feasible search as the complexity is still exponential. Random selection of parameters and exhaustive search has a very positive property. It is possible to determine the global optimum but this comes with the cost of time complexity when aiming to find the global optimum [201].

When relaxing the problem to find local optimums - not guaranteeing to find the global optimum - there exist several computation-efficient strategies. We like to highlight, that applying a local optimum is still better than randomly selecting values. For example, random search provides the best option of hyperparameters by testing multiple randomly initialized hyperparameter settings. This mechanism does not even ensure that one finds a local optimum, but enlarges the probability to find on average a better performing set of parameters than just simply applying a single randomized initiated model. In order to find local optima, more sophisticated techniques need to be utilized, which guide e.g. the random search more intelligently.

In the following, we concentrate on defining an automated hyperparameter optimization technique for IFTM models. The proposed hyperparameter optimization approach is based on genetic programming [84] (see Section 2.3.7) introducing a novel scoring function by us.

6.1 Integration of Hyperparameter Optimization into IFTM Framework

The proposed hyperparameter optimization approach should be applicable to the online scenario, as the anomaly detection is applied in the same way. We therefore

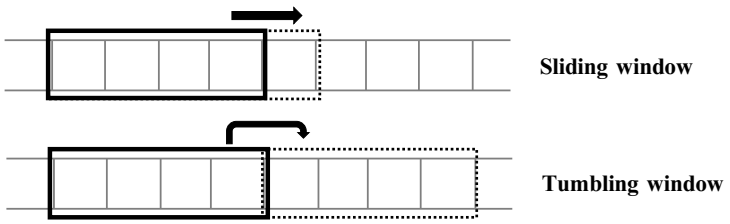


Figure 6.1: Representation of sliding window and tumbling window concepts.

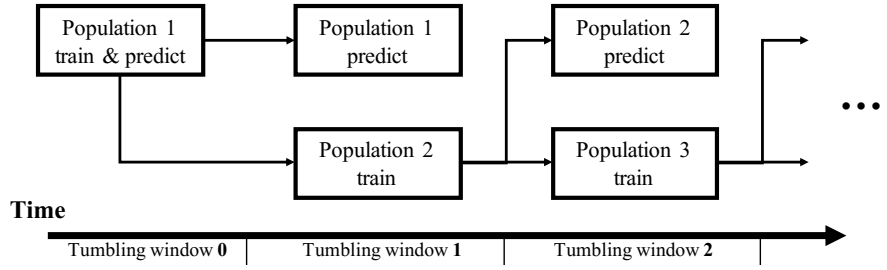


Figure 6.2: Applying training and prediction models to the data stream.

propose the integration of a tumbling window to define the number of steps until a new population is bred. In more detail, a tumbling window represents a set of collected data points. When the set reaches the sufficient size, the window is flushed and collects again new data points. This behavior is illustrated in Figure 6.1 representing the difference to sliding windows. While the sliding window removes the oldest from the queue for each incoming data point, the tumbling window removes the complete window and starts from scratch, which is the aimed approach for determining the points in time to perform the creation of the new breed of population.

Figure 6.2 depicts the different phases in time when populations are point-wise trained and predicted. At first, the models have to be initialized and trained in Tumbling window 0 iteratively as well as point-wise prediction. After reaching the full size of the tumbling window, the trained models are further used for online prediction but are not trained anymore. A new breeding population is trained on the given data stream. Thus, training and prediction are performed in parallel on the same data, but with different models. The models used for prediction are based on the previous tumbling window phase, while the training is performed on the latest models.

Prediction results are based on a population of the latest IFTM models. This population functions as ensembler, providing individual predictions, which are further aggregated to an overall result. Applying multiple ML models as ensemblers is a well-established solution to increase the robustness of the models and lower the number of false alarms [276–279].

For this work, we introduce an ensembler mode for IFTM models as shown in Figure 6.3. Given the set of n IFTM models $S = (m_1, m_2, .., m_n)$, we also have the information about the fitness score for each such model. Thus, we define the aggregated prediction based on the weighted sum of the fitness scores for each model. Based on the performance of the fitness score compared to the other models, the impact of the different IFTM models within a single population may vary and provide therefore different influence on the overall result. The binary decisions of the single models are aggregated, and predicted as normal or abnormal when there are more than 50% of weighted votes (based on the fitness score) from each model are achieved. This decision boundary is configurable to the user and functions as a sensitivity parameter.

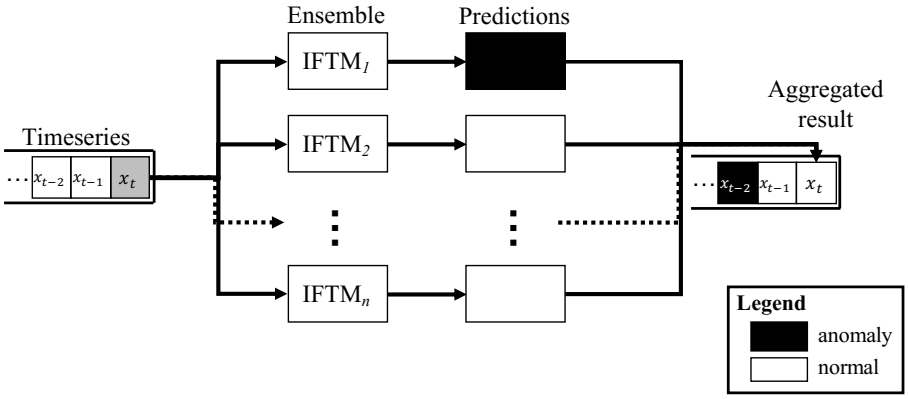


Figure 6.3: Ensemble provides n prediction results, which have to be merged into an aggregated overall prediction result.

Choosing a lower rate to differentiate abnormal reports, results in higher sensitivity when a smaller subset of IFTM models reports an abnormal behavior.

6.2 Automated Hyperparameter Optimization

6.2.1 Initialization, Crossover, Mutation, Termination

Genetic optimization performs a fixed set of methods, which are defined next.

Randomized Initialization and Mutation

At first, an initial population of models is created by randomly choosing hyperparameters (see Appendix B). The random initialization step is performed once at the beginning when creating a novel population of models. Afterward, the crossover phase is applied based on the available population. The crossover consists of a mutation phase, where the randomized selection process is applied to individual hyperparameters when selected for mutation. The mutation takes place with a user-defined probability. New values for parameters thereby have the change to be tested, which may not be available from the crossover phase of initialization.

Parent selection and Crossover

The parent selection as well as the crossover phase are conducted in order to create the new generation of the population (as shown in Figure 6.4). At first, a set of subpopulation is selected from the current population from which the new population for the next iteration is created. The creation is implemented by selecting A models with the highest fitness score in order to create the new population by using the fittest among the population. Furthermore, other B models are selected randomly from the rest of the population in order to provide more diversity of genes (differences in hyperparameters) to the new population. Both of these sets are combined and copied to the new population, but also are used as the parent set from which the crossover is performed in order to generate the rest of the models.

This generation is performed through the crossover phase. For each missing model for reaching again the initial population size, a new model is created on the basis of the parent set. For creating a single new model, two parent models are selected with

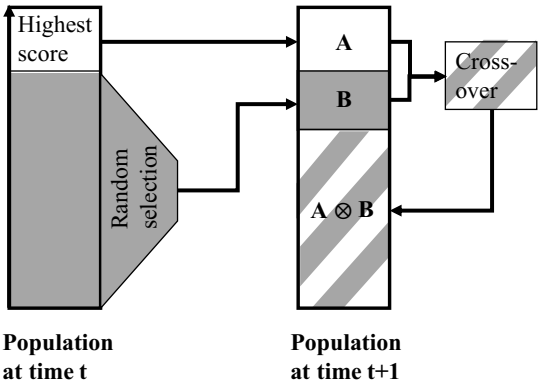


Figure 6.4: Illustration of the crossover from a given population to the next population.

the probability based on uniform distribution from the parent set. Based on these two selected parent models, for every single hyperparameter the parent is randomly (equal probability) chosen and the model parameters are copied. Thus, the resulting new model consists of the hyperparameters from the parents.

Termination criteria

As defined by the Algorithm 1 in step 2, we have to define the termination of the optimization. In general, it is possible to never let the optimization end, but the technique approaches at some point a local optimum. The local optimum is detected when the average fitness score does not significantly change any more. We consider the termination when there is no improvement in the average fitness function score compared to the previous tumbling window at all anymore.

6.2.2 Fitness Function Definition

The fitness function is a crucial component of the optimization algorithm as it defines the optimization criterion on which the hyperparameters are optimized. When this defined criterion is not representative - when this criterion does not correlate with the aimed high accuracy - the optimized set of hyperparameters is not going to perform appropriately as well. A valid candidate is the accuracy of a validation dataset, but this assumes labeled data for normal and abnormal cases. For providing an automated optimization process, the fitness function cannot rely on supervised knowledge and should consequently function under the same constraints of IFTM approaches. The constraints capture the assumption that most data is normal, which provides the option to define the scoring function based on the false alarm rate. When optimizing this criterion, the problem of overestimating the threshold arises. The model detects in this case all data points as normal due to a too large threshold value as the model is optimized towards providing a low false alarm rate.

Further parameters should therefore be investigated to estimate a good fitting model. Figure 6.5 illustrates three different models, which are different in their threshold computation considering that the IF predicts the same reconstruction error (indicated by dots). Threshold 1 does not produce any false alarms but has a large distance to the errors, which could be problematic as anomalies, which produce higher errors, may still remain beneath the threshold and will therefore be misclassified. Threshold 3 is much smaller but produces a high number of false alarms. Threshold 2 produces

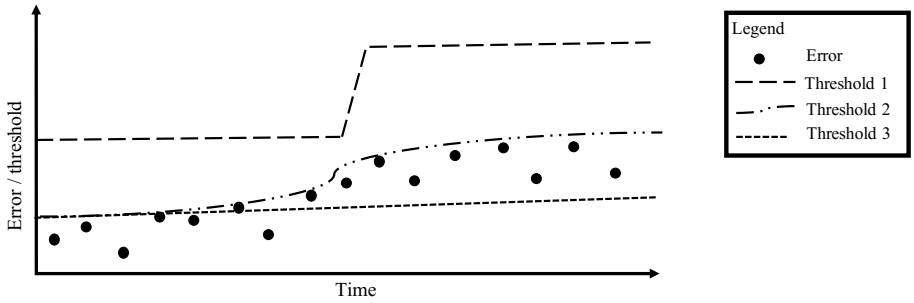


Figure 6.5: Examples of different performing IFTM models.

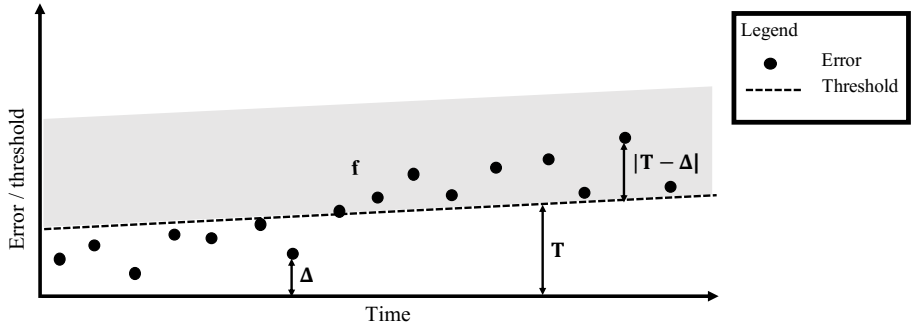


Figure 6.6: Illustration of different indicators for selecting good IFTM models. The dots in the gray area are stated as abnormal data points, consequently increasing the false alarm rate when expected that all such data points are normal.

instead no false alarms but captures precisely normal instances. Thus, small deviations above the threshold would trigger an anomaly, which is the aimed goal. The balance between fluctuations of the reconstruction errors, which are produced through normal data points, and the distance to the threshold has to be optimized. Furthermore, the identity function should be able to provide an accurate prediction for the normal data and therefore low errors are expected.

Based on these properties, we define the compactness of an IFTM model by a fitness score s. For normal behaving signals, we expect that a suited identity function can reconstruct the signal almost perfectly. Consequently, we expect a low reconstruction error Δ. When also given a low threshold T, the model can react to changes and report those as anomalies. The threshold T is also expected to be represented by a small value. Accordingly, the distance between the reconstruction error and threshold value should be small in order to represent compactly the given normal data. The false alarm rate is incorporated and functions as an antagonist. A rise of false alarms is expected as small spikes of the reconstruction error occur when having a small threshold. The false alarm rate f is aimed to penalize too high thresholds. Figure 6.6 presents these four different aspects of f, Δ, T and $|T - \Delta|$ with respect to a given IFTM output.

A number of latest n data points from the last tumbling window are used to calculate the fitness score s as defined by:

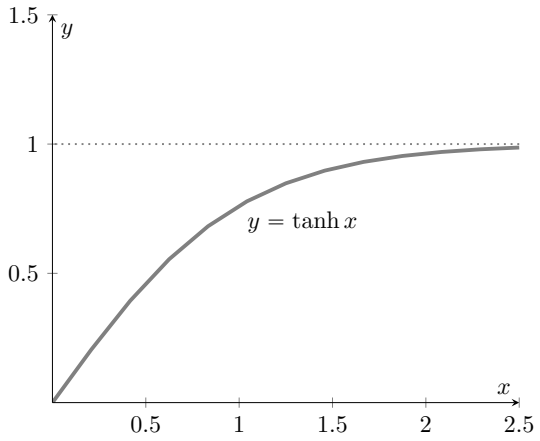


Figure 6.7: Illustration of tanh function in a positive interval.

$$\begin{aligned} argmax\ s = 1 - argmin\ (w_1 \cdot \frac{1}{n}\sum_{i=1}^{n} f_i + w_2 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} \Delta_i) \\ + w_3 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} T_i) + w_4 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} |T_i - \Delta_i|)) \end{aligned} \tag{6.1}$$

The weights $w_{1,\ldots,4} \in [0,1]$ define the importance of the single terms. Thus, user-defined configuration can be applied reflecting the sensitivity of IFTM models, which quickly allow the detection of anomalies, but might rise a higher number of false alarms. In the case of non-sensitive configuration, anomalies might not be recognized. These terms are recommended to be configured as $w_1 = w_2 + w_3 + w_4$ in order to provide a balanced configuration between false alarms and compact fitting to the data stream behavior of the IFTM model. As IFTM models are used to represent the normal behavior, the reconstruction function should approach zero (2. term $w_2 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} \Delta_i)$) as this means that the identity function almost perfectly captures the normal state. Furthermore, the threshold should also be near zero (3. term $w_3 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} T_i)$). This causes a small distance between error and threshold (4. term $w_4 \cdot tanh(\frac{1}{n}\sum_{i=1}^{n} |T_i - \Delta_i|)$), such that the model is highly representative, but might lack a high number of false alarms, as the error might easily get above the threshold. Thus, we also consider to have a low false alarm rate as antagonist for normal data, in the best case also approaching zero (1. term $w_1 \cdot \frac{1}{n}\sum_{i=1}^{n} f_i$). The function $tanh$, as shown in Figure 6.7, is applied to normalize the distances between $[0,1]$ as they shall approach zero. The fitness score $s \in [0,1]$ is aimed to be maximized by the genetic optimization.

6.3 Evaluation

Fitness Function Evaluation The applicability of the defined fitness function (see. Equation 6.1) is shown through conducting an experiment based on the cloud monitoring data as described in Section 7.1. The dataset consists of 952 quantums, which are tested individually and produce their own results. For evaluation purposes, a single quantum is split in separate parts: the first 2:30min of data were used as training set of each quantum, the following 30s of normal data as validation set and the last 2min (1min normal and 1min abnormal data) were used as test data as illustrated in

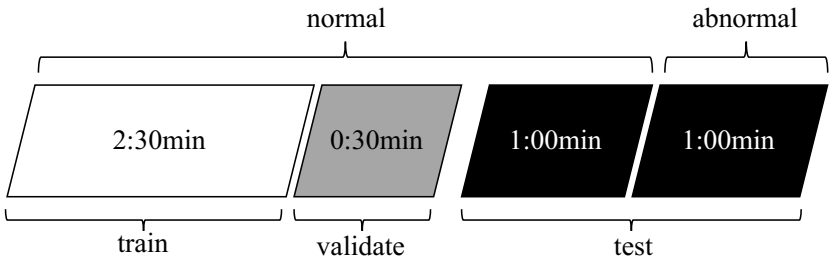


Figure 6.8: Separation of data of a single quantum.

Algorithm	Parameters
CABIRCH	max node entries: [2,20] logistic decay max value: [0,1] logistic decay learning rate: [0,1]
VAE	encoding layers: [1,4] decoding layers: [1,4] learning rate: [0,1]
MArima	mk: [10,200] d: [1,8]
AELSTM	encoding layers: [1,4] decoding layers: [1,4] learning rate: [0,1]
Gaussian threshold	σ: [0,5]

Table 6.1: Table of IFTM model configurations.

Figure 6.8. The separation of the quantum into the three phases of training, validation and test set are based on a time-wise split. The IFTM model trains in the first phases, but it will not be changed for the two upcoming test phases. This enables the evaluation whether fitness score calculations (based on the validation set) reflect the accuracy of the test set.

Furthermore, we applied several different IFTM models based on CABIRCH, Variational autoencoders (VAE), multivariate online Arima (MArima) and an Autoencoder with LSTM cells (AELSTM) as identity functions (see Section 2.3) and Gaussian distribution model as dynamic threshold using randomized initialization through the following uniform distributed hyperparameters as shown in table 6.1. Based on this configuration, a set of 1,000 randomly initiated models for the different IFTM models per quantum is initialized.

Exemplary, we present detailed results through applying first random CABIRCH models. Figure 6.9 shows the Euclidean distance between the reconstruction error and threshold. The image illustrates that models, which have a larger distance in the validation set, represent the same behavior for the normal section of the test set indicated by a Pearson correlation coefficient of 0.9935. The same correlated behavior describes the false alarm rate from the validation set compared to the normal part of the test set with a correlation coefficient of 0.9211 (see Figure 6.10).

Figure 6.11 presents boxplots showing the distribution of reconstruction error for the validation set and test set. The illustration shows that the validation set as well as the first part of the test set show similar distributions, while the abnormal part of the test set shows larger reconstruction errors. This is expected as the training is

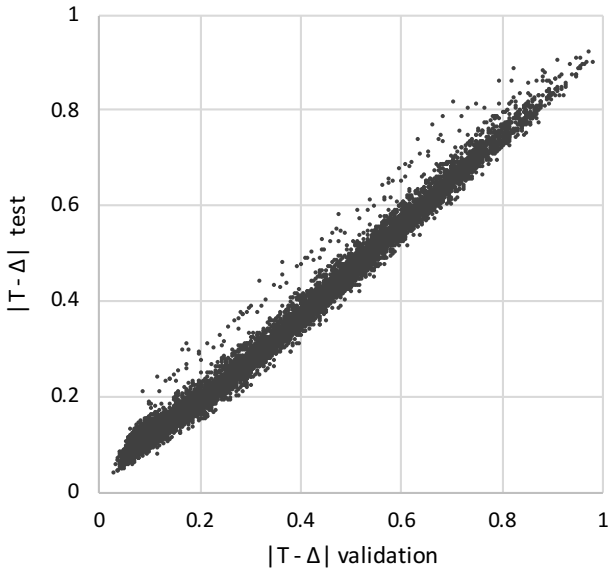


Figure 6.9: Comparison between validation and test set (normal) of Euclidean distance between reconstruction error and threshold. Pearson correlation coefficient: 0.9935.

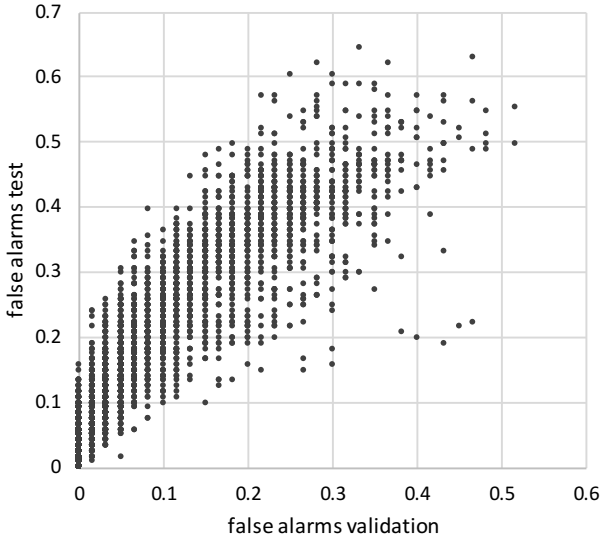


Figure 6.10: Comparison between validation and test set (normal) of false alarm rates. Pearson correlation coefficient: 0.9211.

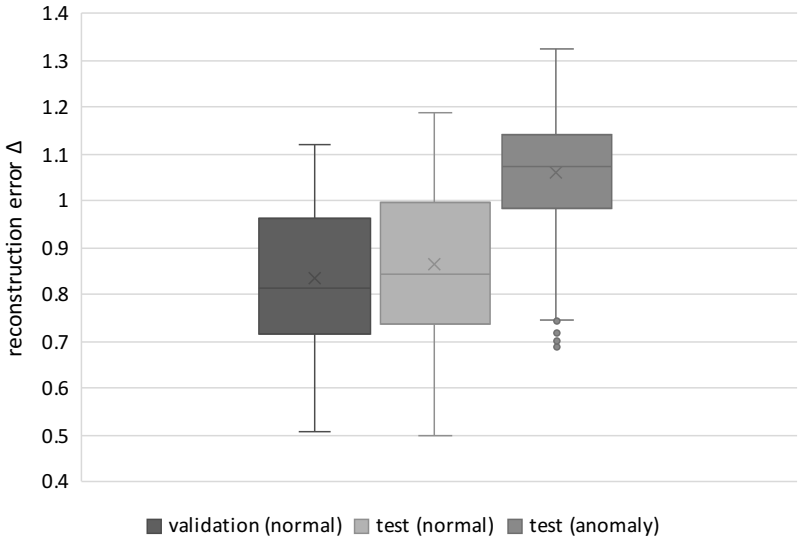


Figure 6.11: Boxplot of reconstruction errors of the validation and both test sets (normal and anomaly).

w_1	w_2	w_3	w_4
0.5	0.125	0.125	0.25

Table 6.2: Weights used for the fitness function.

performed on normal data and should therefore represent more precisely the normal data in the validation set and normal behaving part of the test set.

The behavior of the given data is therefore expected and indicates, that the time horizon of prediction does not underlie too much change in the normal state in the validation set and normal test set, while the abnormal test set shows higher reconstruction errors.

Table 6.2 shows the individual weightings of the fitness function. Half of the score value is contributed by the false alarm rate, while the other half is defined up to the three further score's metrics. This is intended to weight the antagonists equally as suggested above. The rest of the three values are also split up into two groups. While w_4 represents the distance between threshold and reconstruction error is weighted as high as the actual largeness of the threshold and reconstitution error, resulting in the weights is presented in Table 6.2.

Figure 6.12 presents the dependency between the distance of the reconstruction error and the threshold compared to the false alarm rates for the anomaly case and normal case using the test set data. The black points represent the normal data of the test set, where a low distance between the reconstruction error and threshold (here after referred to as distance) shows an increase in the false alarm rate, while a large distance shows a preferred low false alarm rate (near zero). This is expected as a large distance compensates too much the fluctuations in the error and therefore all data points will be detected as normal. A small distance in contrast shows higher false alarms as the model may over-fit the signal and does not compensate the noise in the error.

The brighter points illustrate the dependency between the distance with the ab-

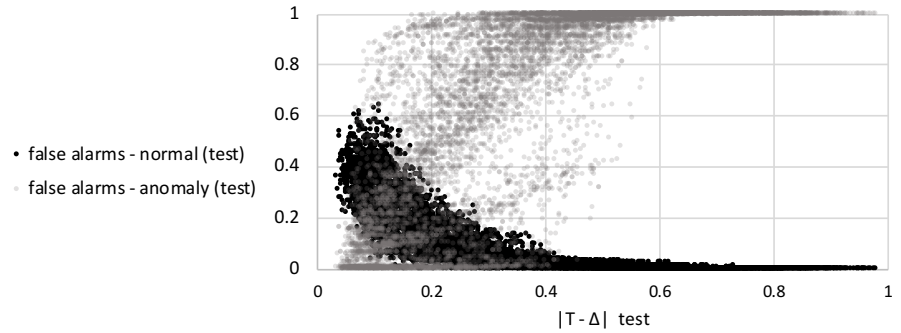


Figure 6.12: Illustration of false alarms within the normal and anomaly test sets compared with the distance between error and threshold.

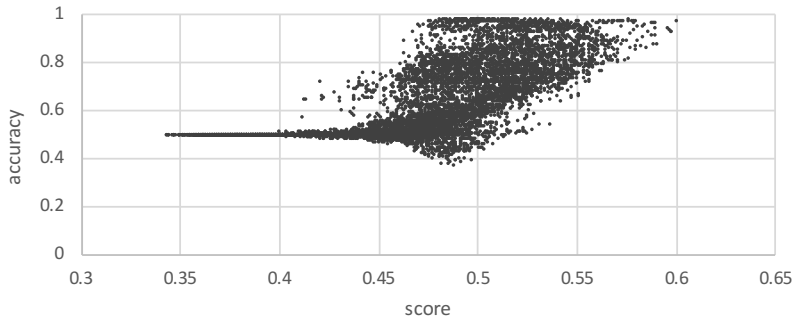


Figure 6.13: Scatterplot showing the comparison of the score based on the validation set and resulting accuracy of the test set. The resulting Pearson correlation is: 0.73.

normal data, showing that small distances indicate a low false alarm rate and higher distances misclassify the anomalies more, represented by a high false alarm rate for the abnormal data. This is again expected, as a high distance is going to classify data as normal leading to high false alarms for abnormal data, while small distances provide the chance to jump above the threshold causing a trigger for an anomaly. Figure 6.12 presents the problem of choosing a robust model, which provides a balance between a small distance and the false alarm rate.

The scatter plot (Figure 6.13) shows the relationship between calculated scores based on the validation set and the accuracy based test set. The plot shows, that there exists almost a nonchanging behavior of accuracy of 0.5 for scores below 0.4 representing models, which are able to just detect abnormal or just normal behavior and those which produce perfectly balanced misclassifications. Higher larger scores of the fitness function show an increase in accuracy. This results in a correlation coefficient of 0.73 using the Pearson correlation for the complete dataset.

The above described figures showed detailed behaviors for a CABIRCH based anomaly detection model. Table 6.3 describes the overall correlation in the last column C(s, a), where a denotes the accuracy, s the score and C the correlation function applied, for the different IFTM models (MArima, VAE, AELSTM). All those models show high correlations with respect to the score and accuracy. In addition, the table presents the correlation of the individual values used within the fitness function with

	$\mathbf{C}(\Delta, a)$	$\mathbf{C}(T,a)$	$\mathbf{C}(\lvert T-\Delta\rvert,a)$	$\mathbf{C}(f,a)$	$\mathbf{C}(s, a)$
CABIRCH	-0.45	-0.73	-0.65	0.34	**0.73**
MArima	-0.28	-0.44	-0.45	-0.04	**0.70**
VAE	-0.03	-0.07	-0.07	0.14	**0.68**
AELSTM	-0.08	-0.09	-0.09	0.19	**0.65**

Table 6.3: Pearson correlation of individual scoring metrics with respect to the accuracy for different IFTM models.

Parameter hyperoptimization	**Default values**
Mutation rate	5%
Tumbling window size	100
Initial population size	100
Parent selection: Best percentage	20%
Parent selection: Random percentage	20%
Fitness score weightings w_1, w_2, w_3, w_4	$w_1 = 0.5$ $w_2 = w_3 = 0.125$ $w_4 = 0.25$

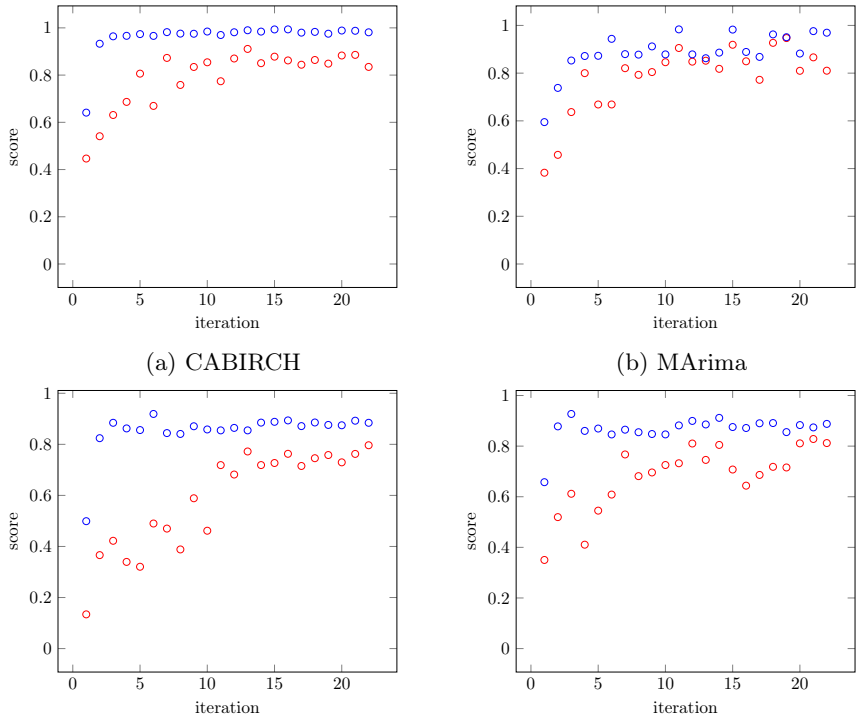
Table 6.4: Default values for the hyperoptimization for IFTM models.

respect to the accuracy. For each of the individual metrics, there does not exist a high correlation for any of the applied approaches. This reflects that the individual metrics do not provide enough information to approximate the accuracy a model can reach. Therefore, the combination of the individual metrics through the fitness function is necessary and applicable.

Cloud Monitoring Evaluation Table 6.4 presents the configuration of our hyperparameter approach. Combined with the IFTM model specific configurations (see Appendix B), anomaly detection is applied to the cloud monitoring dataset (see Section 7.1).

The four plots of Figure 6.14 show the results when applying the different IFTM models (CABIRCH (a), MArima (b), VAE (c), and AELSTM (d) as described above) to the sequence of 20min of the initial normal load of the monitoring data. The plots represent the behavior of the population's score developing over time. After each tumbling window (described as iterations - x-axis of the plots), the average score (red) and the average of the best performing 20% of scores (blue) are reported. Each IFTM model was applied 100 times to provide representative behaviors to the randomized internal processes of the hyperparameter optimization. The diagrams show an increase in both scores over time for all individual models. The best scores reach high scores (above 0.8) in a short period of time (ca. 5 iterations), while the overall population's average approaches high scores later in time and depend on the applied IFTM model. CABIRCH and MArima show in the average case (red) quicker response to increasing scores, while the neural network-based approaches increase slower. All plots illustrate a converging behavior to improving scores over time, which reflects the effectiveness of applying hyperparameter optimization.

In the previous Chapter 5, the optimal result for CABIRCH was determined through grid search with respect to the AUC. Table 6.5 presents these results and adds the results when applying hyperparameter optimization-based CABIRCH. The



(a) CABIRCH (b) MArima

(c) VAE (d) AELSTM

Figure 6.14: Behavior of score development over time. Blue: Average of 20% population's highest scores. Red: Average over the complete set of population's scores.

Algorithm	**TP-rate**	**TN-rate**	**TP-event**	**AUC**
Hyperopt CABIRCH	56.04%	85.82%	94.84%	70.93%
Best CABIRCH	54.27%	96.69%	96.23%	75.48%
Difference	+1.77%	-10.87%	-1.39%	-4.55%

Table 6.5: Comparison between hyperparameter optimized model and overall best CABIRCH model.

table shows a drop of 4.55% in AUC value, which relates to the large difference in representing the normal state. This is represented by the TN-rate, where the hyperparameter optimized version provides 10.87% more false alarms (14.18% false alarms in total), while detecting 1.77% better anomalies. Reflecting the production near parameter for detecting anomaly events, the TP-event rate is decreased by 1.39%.

Applying hyperparameter optimization to CABIRCH combined with the Gaussian threshold model leaves room for improvements due to its high TN-rate. In the following Chapter 7, we compare 27 combinations of identity functions and threshold models investigating the potential for unsupervised anomaly detection when applying CABIRCH and further state of the art techniques as identity functions together with hyperparameter optimization.

Chapter 7

Evaluation

For elaborating unsupervised anomaly detection for detecting online abnormal behaviors in cloud monitoring data streams, we apply 28 IFTM algorithms, including state of the art anomaly detection methods. IFTM generalizes the applicability between such reconstruction or forecasting methods as identity functions and enables applying different types of dynamic threshold models.

Table 7.1 presents all integrated IFTM approaches, seven identity functions and four threshold models.

Besides the proposed anomaly detection based on CABIRCH and Gaussian threshold modeling (CA), we apply further dynamic threshold models: Sliding window aggregation (SWA) , Exponential moving model (EMM) [69] and Double exponential moving model (DEMM) [68]. In addition, we evaluate the further identity functions: Autoencoders (AE), Variational Autoencoder (VAE), Univariate Arima (UArima), Multivariate Arima (MArima), LSTM-based network (LSTM) and Autoencoder with LSTM cells (AELSTM). For all of these approaches, we provided details in Section 2.3.1 and in case of Arima in Appendix A.

Laptev et al. [147] proposed to apply Arima together with a Gaussian threshold for univariate time-series anomaly detection. Due to applying online learning, we utilized an optimized form of Arima, called Online Arima, and showed its applicability with a Gaussian threshold to the cloud monitoring domain [268, 269]. Whether further improvements are possible with other threshold models, is evaluated in this thesis.

Hundman et al. [65] applied an LSTM network as identity function together with EMM as threshold model to multivariate time-series anomaly detection for mars rover data. In contrast, Malhotra et al. [66] utilized a Gaussian threshold, while applying also an LSTM network, and showed its applicability to a wider range of domains like ECG signal analysis, engine sensor data and power consumption monitoring. In both cases, they did not investigate the applicability to cloud anomalies.

Oh and Yun [67] illustrated the applicability of AE and Gaussian threshold as a key model for machine sound signal anomaly detection, while Xu et al. [176] applied a VAE together with a Gaussian threshold to analyze web application KPIs. Again, both of these papers did not evaluate the applicability to cloud monitoring data and did not investigate further dynamic threshold choices.

AELSTM combines the structure of AE with employing LSTM cells. Such a structure was proposed by Park et al. [173] to detect abnormal behaviors in robot-assisted feeding settings, but applied in a semi-supervised manner without automatically inferred thresholds.

Based on these findings of recent work in unsupervised anomaly detection, we integrated all these different methods into IFTM. The neural network architecture of integrated autoencoders (AE, VAE, AELSTM) creates for each layer a decreased

Identity function	Threshold model
CABIRCH Autoencoder Variational Autoencoder Univariate Arima Multivariate Arima LSTM network Autoencoder with LSTM cells	Gaussian aggregation Sliding window aggregation Exponential moving model Double exponential moving model

Table 7.1: List of integrated IF and TM models in the IFTM framework.

number of neurons by $n_{i+1} = \lceil \frac{n_i}{2} \rceil$, where N_i are the number nodes from the previous layer i, for the encoding network. The decoding network is symmetrically copied and increases with the number of neurons. The network structure for LSTMs as an identity function is kept straight forward: the number of nodes in the input and output layer are the same as the number of dimensions given through the multivariate data point. Further hyperparameters are optimized based on out optimization approach with the base configurations seen in Table 6.4 and the interval selections presented in Appendix B.

7.1 Evaluation Setup

For evaluation purposes, we built an extensive private cloud testbed running the cloud services Clearwater and Video-on-demand.

Cloud Infrastructure

The infrastructure used within these experiments are based on a homogeneous cluster setup of 13 physical servers, where each server has the following properties:

- Intel Xeon X3450 (4 cores @ 2.66GHz)
- 16GB Ram
- 3x 1TB disks
- 2x 1GBit Ethernet interfaces

On these servers, a highly available OpenStack installation is deployed, which functions as private cloud [280]. Since OpenStack consists of management services itself, OpenStack occupies a subset of physical hosts. Thus, 9 physical hosts are used as compute nodes for evaluation. Through OpenStack, it is possible to create virtual machines through KVM as virtualization engine, in which we deploy the services described above. For each service installation, we created its own virtual machine. The virtual machines run Ubuntu 16.04 and use 2 vCPU cores, 2GB memory and 20GB disk.

Clearwater

Section 2.2.1 described the open source Project Clearwater and its internal components. Based on these components, we deployed a replicated version which is load balanced. The key services Bono and Sprout are deployed three times each, while single deployments are used for Homer, Homestead, and Ellis. In order to provide realistic results, we simulate the usage of the IMS by changing the number of client registrations and call initiations randomly between 20 and 40 users every minute.

The system was monitored under normal load behavior for the first 20 minutes. Afterwards, the anomaly injections started injecting with a duration of 5 minutes followed by a cooldown phase of 1 minute normal data. Execution of anomalies is switched round robin, first through the hosts and then the anomaly type.

The complete execution plan runs for 96h. Data was collected with a monitoring interval of 500ms and collected in parallel on all the given service components. In order to provide a clean dataset for evaluation, phases when anomaly executions run on different hosts are removed as they may influence the result due to propagation, which is out of scope for the anomaly detection and referred to the root cause analysis. The resulting dataset for each monitored service consists of 286,108 data points containing 24 resource metrics.

Video-on-demand

Section 2.2.2 describes the Video-on-demand service use case. A replicated version of the load balancer and video backend servers are deployed, with four clients producing the service load. The distribution upon the physical servers has been carried out through the different parts of the service to consequently distributed among three servers.

The system was monitored again under normal load behavior for the first 20 minutes. Afterwards, the anomaly injections started injecting with a duration of 1 minute followed by a cooldown phase of 4 minutes normal data. Execution of anomalies are switched round robin, first through the hosts, and then the anomaly type.

The complete execution plan runs for 96h. Data was collected with a monitoring interval of 500ms and collected in parallel on all the given service components. In order to provide a clean dataset for evaluation, in phases when anomaly executions run on different hosts, they are removed as they may influence the result due to propagation, which is out of scope for the anomaly detection and referred to the root cause analysis. The resulting dataset for each monitored component consists of more than 280,000 data points containing 24 resource metrics.

7.1.1 Resource Monitoring

The Bitflow collector[1] is applied for monitoring the cloud infrastructure on different layers. The collector is implemented in Go and aims to monitor resource metrics in user-defined frequent intervals, which are sent through a TCP connection in a unified format to a given destination. The Bitflow collector retrieves monitoring information from different interfaces, which are mainly from the proc-file system, libvirt [281], and ovsdb [282].

Table 7.2 shows the different layers and metrics on which the collector can aggregate information. The left column shows the different types of metrics collected and the following columns define on which level those are collected. The proc-filesystem is being used for the system-wide and process-wise collection of metrics, while libvirt is used for virtual machine monitoring from the hypervisor level and ovsdb for further information of the network layer.

In addition, preprocessing is conducted by the monitoring agent. Based on historic data it applies min-max scaling:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{7.1}$$

Where x' denotes the normalized value and x_{min} the minimum value and x_{max} the maximum value of a given metric.

[1] `https://github.com/bitflow-stream/go-bitflow-collector/`

Metric	Description	system-wide	per process	libvirt per VM	ovsdb per bridge
CPU	CPU utilization	x	x	x	
RAM	RAM utilization	x	x	x	
Disk IO	Hard disk access in bytes and times	x	x	x	
Disk space	Disk space usage per partition	x		x	
Network IO	Network utilization in bytes and packets	x	x	x	x
Network protocol	Protocol specific counters for IP, UDP, TCP, ICMP	x			
Num. of processes		x			
Num. of threads			x		

Table 7.2: Interfaces of resource metrics monitored by the Bitflow collector.

	Successive increase	Rapid change	Fluctuations
CPU	x	x	x
Memory	x	x	x
Hard disk	x	x	x
Network		x	

Table 7.3: Types of degraded state anomalies compared to aimed metrics.

7.1.2 Anomaly Injection Framework

For simulating degraded state anomalies, we developed an injection framework. It consists of three main components: the experiment controller, the load controller, and the injection agent as depicted in Figure 7.1. The experiment controller is a centralized service, which is deployed once and manages triggering of anomalies. Likewise, the load controller is a centralized component for managing the load changes within the system.

The load controller component contains a SIP load generator module, which initiates sessions for a certain number of users per a defined temporal period for the Clearwater IMS system. This load generator is a sip-stress component provided by Clearwater itself, which is deployed within a separate virtual machine and used to generate defined system loads. For the Video-on-demand VNF service, the same principles are deployed, where the load controller initiates the streaming of videos through clients simulating users.

The anomaly injection agent is deployed inside each component, which is monitored and where the user wants to execute anomalies. The agent is controlled through a RESTful API, providing the options to trigger or stop anomalies, set anomaly execution time plans, or get the status of all or certain specifically injected anomalies.

The integrated anomalies execute through dedicated processes the simulations of different resource usage patterns based on the defined patterns in Section 3.1. These provide different stationary change types within different metrics like CPU, memory, disk and network. The main differences are realized to provide either instant changes, which are run by the Stress-ng service[2], or by slowly progressing anomalies like memory leaks. Furthermore, there are also differences in either constant behavior of the anomalies versus fluctuating anomalies. Table 7.3 summarizes the main differences in type compared to the metric types included in the anomaly injection framework.

[2] `https://wiki.ubuntu.com/Kernel/Reference/stress-ng/`

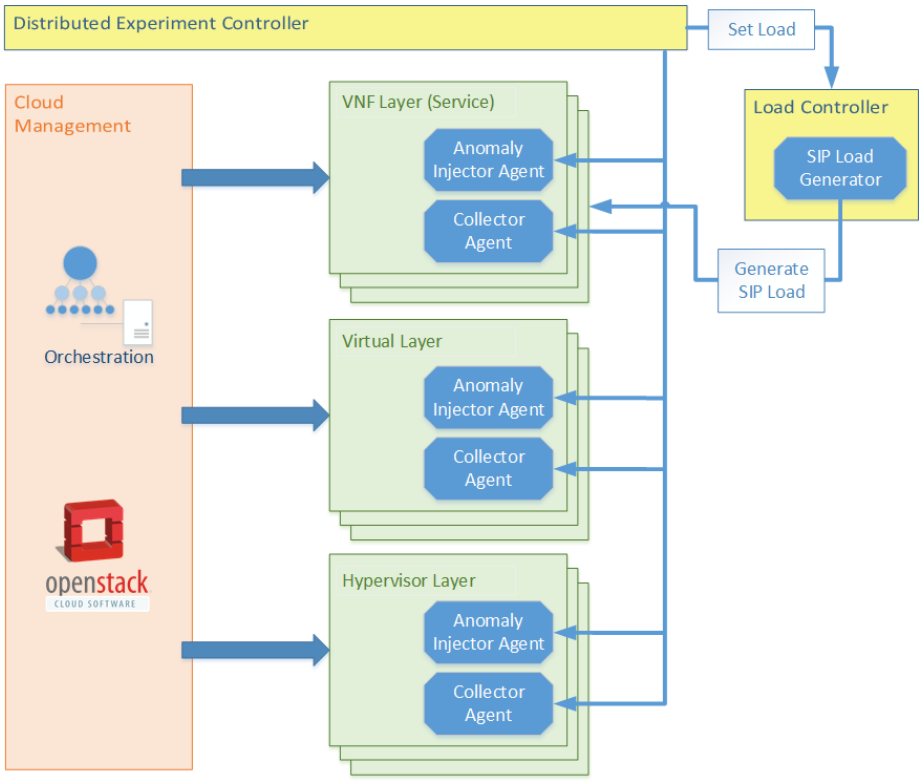


Figure 7.1: Distributed experiment controller and placement of injection agents and collector agents.

The complete list of integrated degraded state anomalies consists of:

1. *Disk pollution, temporary disk pollution* are both writing data into a log-file. For the temporary case, the files are deleted after execution, while in the other case the file will still be available.
2. *HDD stress* describes an excessive hard disk utilization by writing a file. Stress-ng spawns between 1 and 4 workers executing this task.
3. *CPU stress* immediately consumes excessively much CPU. *CPU leakage* describes an anomaly increasing in its intensity over time by consuming more and more CPU. The *fluctuation of CPU* constantly increases and decreases the CPU usage in order to provide a fluctuation behavior.
4. Like the CPU, memory anomalies are also modeled with the same behaviors: immediately consuming high amounts of memory (*memory stress*), growing consumption over time (*memory leakage*) and fluctuating allocation of memory (*memory fluctuations*).
5. For the anomalies *fork flooding leak* and *fork flooding fluctuation*, a process starts the creation of child processes over time. For the leakage variant, those child processes are again creating more processes, while in the fluctuation variant children are also removed partly.
6. *Large file download* applies downloading a large file, resulting in a high network traffic.
7. *File pointer wasting* requests file pointers without releasing them, creates a leakage of the pointer IDs over time.

7.2 Evaluation Results

Figure 7.2 presents the TP-rate for the detected anomaly events and TN-rate for describing the portion of false alarms. Most algorithmic combinations (19 out of 28) produce a 100% TP-event detection rate, which is the highest value, while 9 do not detect all the given events. The TN-rate is shown, indicating further differentiation. Whether the model is appropriate to capture the normal state is reflected by the TN-rate. Consequently, the preferred approach reaches high rates for TP-even as well as for the TN. The best options are shown by combining AE & DEMM achieving 100% TP-event rate and 96.14% TN-rate.

Events might not be detected directly when they are initiated. This is expected as anomalies like memory leaks start with very little change at the beginning and grow slowly over time. Due to the small changes at the beginning, it is hard to differentiate to normal changes in the signal. In Figure 7.3, we show the times for this detection delay for these approaches achieving 100% TP-event rates (for the rest see Figure C.2). As the monitoring interval is 500ms, at the first or second monitored instance most of the detection approaches alert the anomaly correctly. There exist still further techniques utilizing a few seconds in time in order to detect the anomalies correctly. This behavior is especially visible for IFTM models achieving higher TN-rates. This can be explained due to compactness of the model as those which reach a smaller number of false alarms provide larger distances between the prediction error and threshold. Consequently, the chance of detecting small changes is smaller.

The correlation between average times and the standard deviation correlates for the anomaly events as shown in Figure 7.4. The correlation results in a high Person correlation coefficient of 0.93 for the abnormal detection times, but as well for the normal false alarm phase duration with 0.94. This behavior was also shown in Chapter 5 (see Figure 5.17). The complete and detailed list of combination's correlations is provided in the Appendix C Figure C.3.

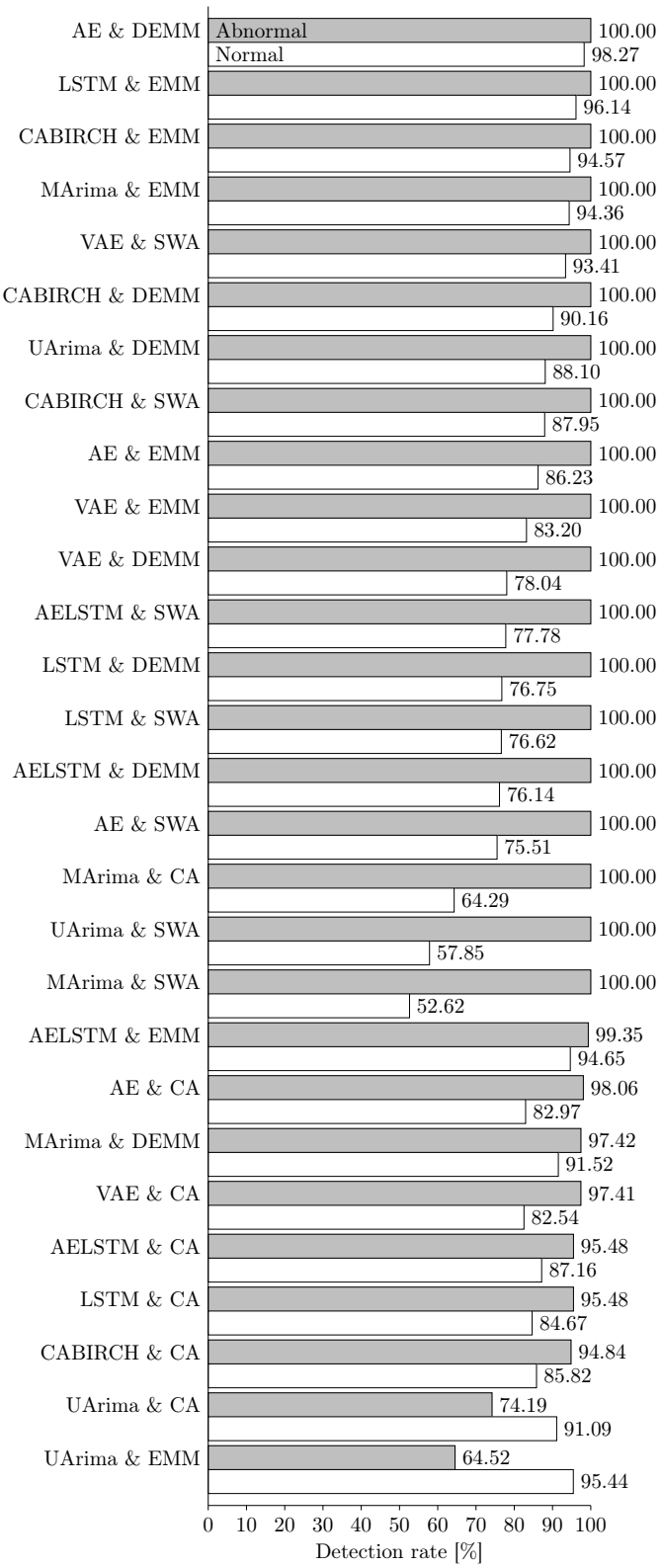


Figure 7.2: Anomaly event detection rate (TP-event) and TN-rates for all 28 IFTM models.

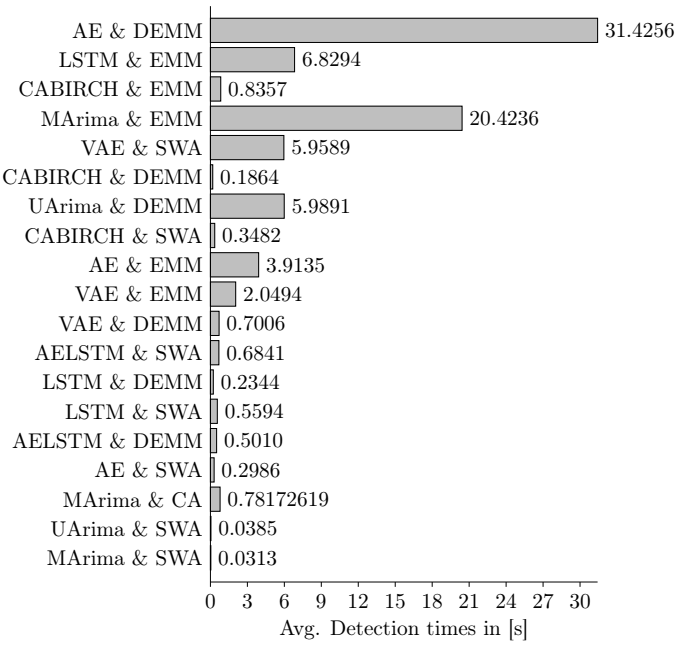


Figure 7.3: Avg. Detection times in [s] for all above 100% detection rate.

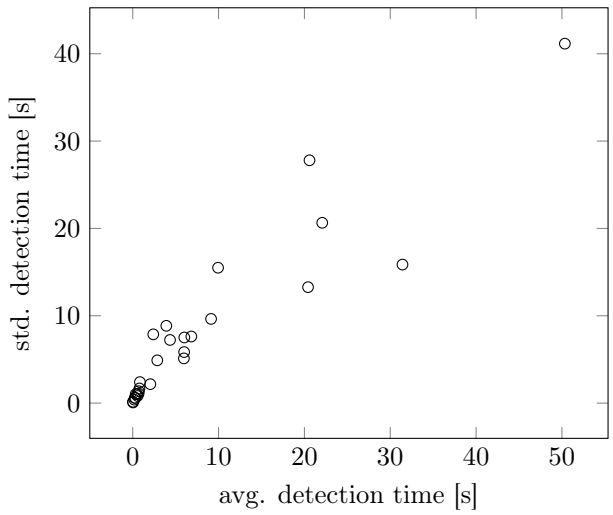


Figure 7.4: Comparison of average detection times and standard deviation of detection times.

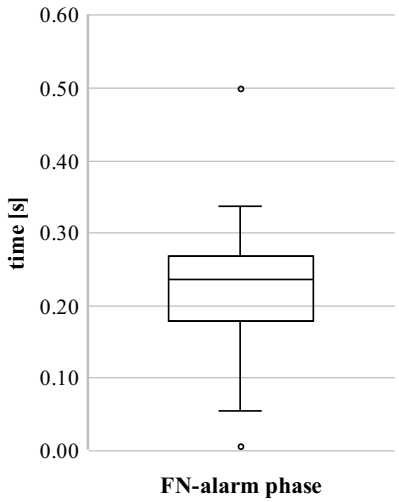


Figure 7.5: Normal false alarm duration.

In case of false alarms, the duration (time in seconds) of those phases is represented in Figure 7.5 as a box-plot. The plot is based on the individual results for all 28 combinations of IFTM models. Individual results per IFTM combination are shown in Appendix C in Figure C.1. As the individual results show similar behavior, we show here an aggregated diagram. The results illustrate small false alarm duration of at most half a second. This means, that on average the false alarm duration consists of between one and two misclassified samples.

When comparing the quality of the given 28 different combinations of IFTM models, we first have to define what the best-aimed approach should be capable of. As described above, the approach should be capable of detecting all anomaly events. Therefore, we show next just those with 100% event detection rate. The rest of the aimed achievements can be described by the following three metrics. At first, the TN-rate should be as high as possible, which means that the false alarms within the normal signal are close to zero. Secondly, the computation time for a single sample should be as low as possible (when aimed at a resource limit of 3% and a monitoring interval, the result should run at most 15ms). The last metric represents the detection delay describing the average time needed until an anomaly alert was proposed.

Figure 7.6 presents these three key indicators, from which the reader can choose appropriate algorithms from its individual requirements. The y-axis presents the TN-rate and the x-axis captures the runtime. The 19 different combinations are presented as dots, showing the dependency between those two metrics. The numbering of the 19 approaches is based on the TN-rate, helping to distinguish the differences. The results show, that there exist several approaches with a small runtime (below the aimed 15ms), but also a high diversity in the TN-rate. Thus, there exist the approaches 3, 4 and 6 with above 90% TN-rate and lower runtime than 15ms. When including additionally the detection delay of anomaly events, we can provide even more differentiation between the quality of results. This is presented as well in this plot by providing a coloring based on the delay time, from near zero (blue) over yellow (ca. 10s) to 30s (red).

As it is problematic to visualize the differentiation, as a lot of values are presented as blue dots, in Figure 7.7 we show the same three metrics from a different angle. This time, the x-axis represents the detection delay for the anomaly events, while the y-axis represents again the TN-rate. As for colors, the runtime is represented. The plot shows

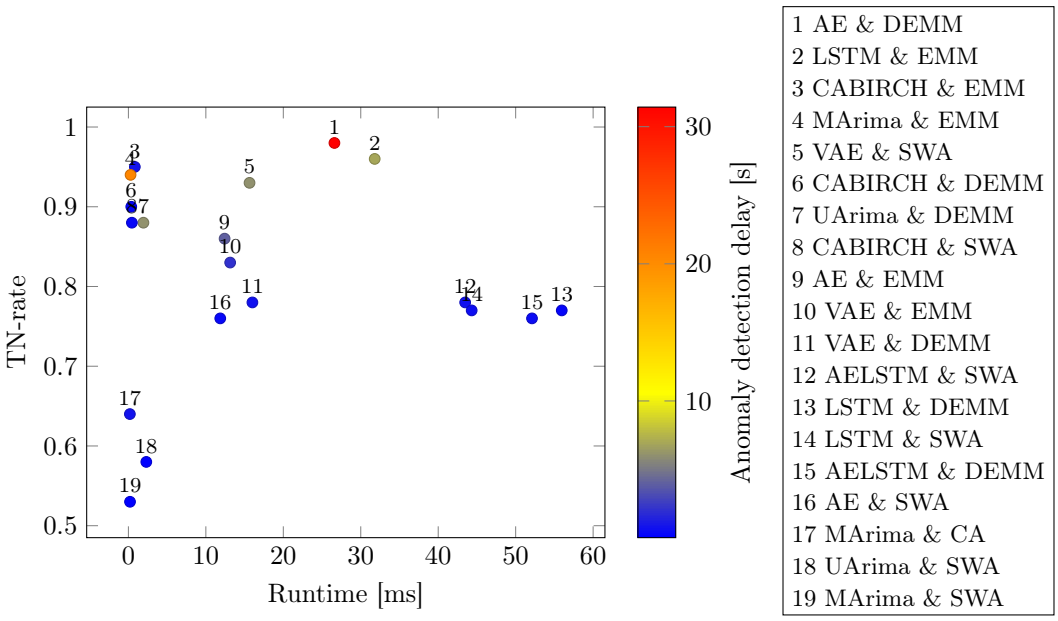


Figure 7.6: Relationship between TN-rate and delay for average runtime per sample for all IFTM models achieving 100% event-detection rate.

that the approach AE & DEMM with the highest TN-rate also produces the highest delay of a little more than 30 seconds on average. The plot also illustrates, that with decreased TN-rate, the time of the delay reduces as well. This is expected as these models provide a higher number of FPs and consequently the chance of excellently detecting the anomaly event increases. Such details about the point-wise detection results are presented in Table C.1 and Table C.2 in Appendix C. Six IFTM models achieve above 90% TN-rate, but the models' AE & DEMM and LSTM & EMM reach more than 30 seconds and 20 seconds detection delays respectively, which might not be acceptable depending on the complexity of remediation actions. On the other hand, a single anomaly event runs for up to 5 minutes. Consequently, when assuming that the anomaly would end up into a failure, the selection process and execution of remediation has still more than 4 minutes.

Combining these two plots, Figure 7.8 illustrates a 3D plot of the given three key indicator metrics. The axes accordingly represent one of the three metrics, while the color of the point describes the detection delay to provide additional help to show the three dimensions in a more readable way. The same findings hold as described above. While the IFTM model AE & DEMM shows the highest TN-rate, there exist more efficient models with respect to the computation time and anomaly event delay. The perfect model would have a TN-rate of 100%, near-zero computation time and near-zero detection delay. Encountering this and the requirements of the AIOps platform, we recommend to apply one of the three combinations:

- Option 1: AE & DEMM show the highest TN-rate, but also the highest detection delay and the computation time might be too high with respect to given resource limits.
- Option 2: LSTM & EMM show a significantly lower detection delay, but the computation time increased by 4 seconds. Still, the TN-rate is at the second-highest.

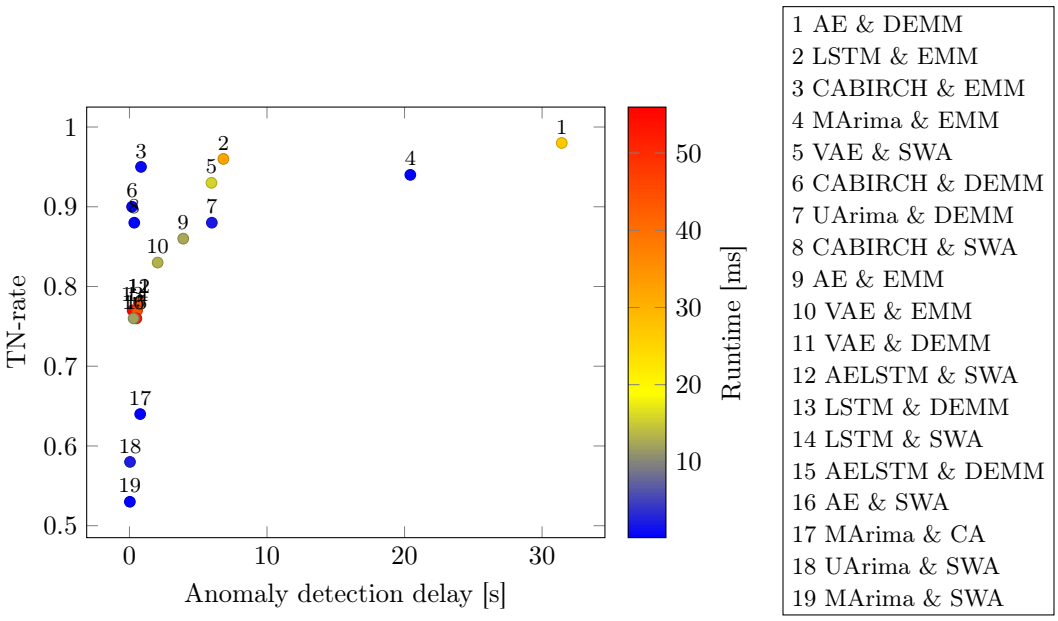


Figure 7.7: Relationship between TN-rate and delay for detecting an anomaly event for all IFTM models achieving 100% event-detection rate.

- Option 3: CABIRCH & EMM show almost zero detection time and almost zero computation time, but the false alarms in the normal signal increased to more than 5%.

There exist further models, which provide better results in one of the metrics and therefore lie on the paretofront, but do not provide relevant impact compared to those three options.

Besides the overall anomaly event detection rates for a given model, we show in Figure 7.9 how accurate the individual anomaly types were detected throughout all the given approaches. Thus, the bar plot shows in its y-axis the TP-event rate as average over all the detection approaches. The results show, that the anomaly type *Bandwidth stress* is detected by all 28 approaches, which indicates that this anomaly is easy to detect. On the other hand, the anomaly type *File pointer wasting* has the lowest rate, below 90%, and indicates the most problematic event to be detected.

Lastly, Figure 7.10 shows the computation time for the IFTM combinations separated by IF and TM. The bar plot shows, that all evaluated deep learning models show more than 10ms computation time per sample, but when using LSTM cells, the computation time increases to more than 30ms. In comparison, the runtimes of UArima, MArima, and CABIRCH are far below 10ms computation time. Furthermore, through the discussed optimization in the Arima computation, the multivariate version computes samples more efficiently than the univariate version. This can be caused due to the capturing of linear interdependencies between the monitored metrics, resulting in a decrease in the size of capturing large internal windows or high differentiation depth. These white box details about the Arima models should be investigated in future work.

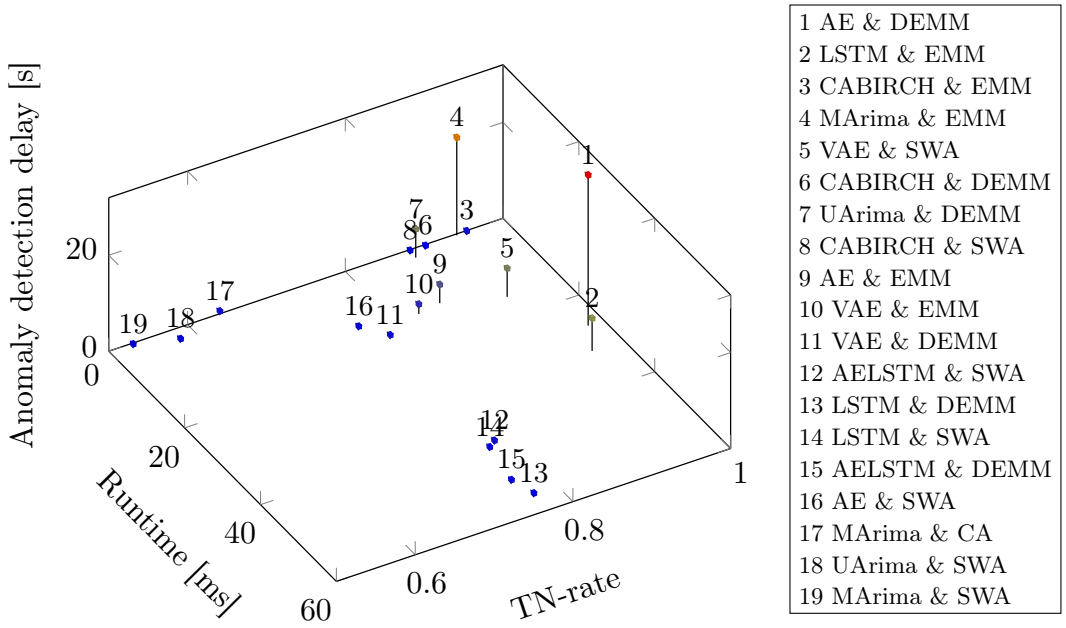


Figure 7.8: 3D scatter plot illustrating the three dimensions of TN-rate, average execution runtime per sample and average detection time after the anomaly event started.

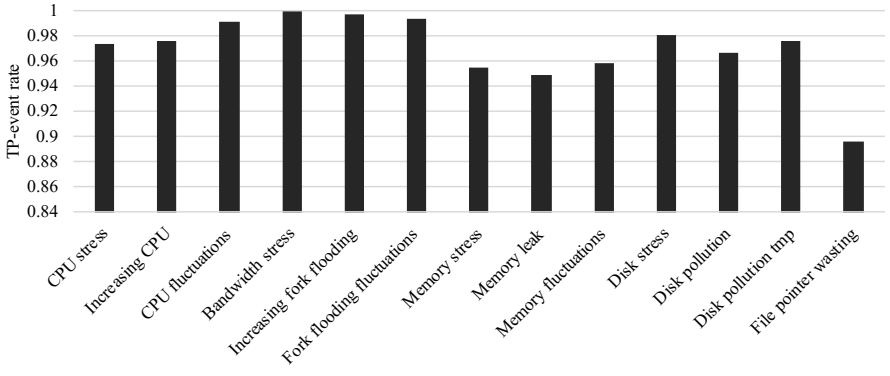


Figure 7.9: Results of TP-event rate per anomaly type.

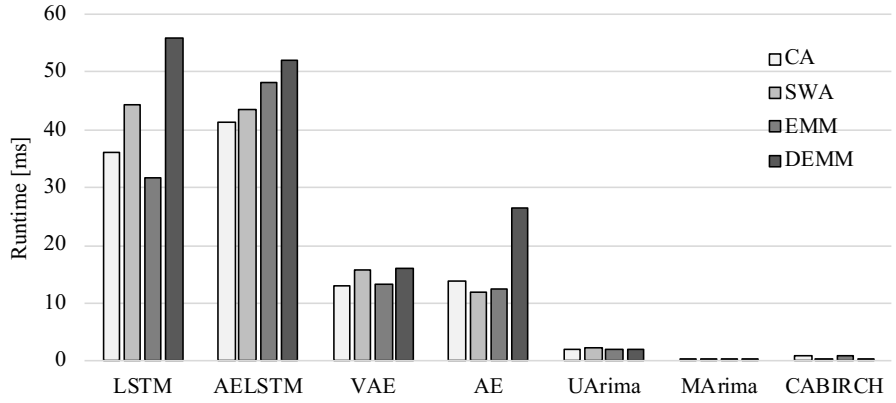


Figure 7.10: Runtimes seperated for IF and TM.

7.3 Discussion

Summarizing aimed needs and requirements for applicable anomaly detection for cloud computing environments are:

1. Near-zero human labeling: Human labeling of data arrives with costs, time, and requires high expertise for the individual monitored services. The solution requires to bound the labeling of data by humans, which also increases the automation process of the overall ZerOps platform.
2. Near-zero configuration efforts for applying ML models: DevOps engineers and administrators of cloud computing environments are in general no ML experts, but need to apply ML models in order to keep companies being profitable. Users should therefore be able to apply ML models without extensive ML knowledge like approach specific hyperparameter selection.
3. Near-zero learning time: When applying AI-based approaches, the key challenge remains the computation complexity for training a model (for most types of ML algorithms).
4. Accurate detection results: Industry demands 99.999% in accuracy [3].
5. Just-in-time processing: Computation time for a sample should be less than the monitoring interval to ensure just-in-time computation for the continuous data stream.
6. Online (point-wise) predictions: The anomaly detection should provide an individual recommendation for each recorded data point, whether an anomaly appeared.
7. Detect anomaly as soon as it appears: The detection mechanism should be able to detect abnormal behavior as soon as it appears in order to provide the chance for the remediation engine to apply suitable actions. Detecting anomalies too late may impact the service quality or causes a failure of the component without the possibility to mitigate the anomaly.
8. Detect unknown anomalies: The anomaly detection should be also capable of detecting unforeseen problems, thus also handling unknown anomalies.

These requirements provide a broad overview of the desired properties of the anomaly detection algorithm. For this evaluation, we utilized unsupervised models. Due to the construction and design of IFTM models, no human labeling is required as it runs unsupervised. Consequently, the 1^{st} property is fulfilled. Additionally, the

IFTM model provides by design an internal border definition by utilizing dynamic thresholds. This helps to differentiate between the normal system state and abnormal behavior. Thus, the 8^{th} property is captured when assuming that anomalies do not appear for long periods of time, and too often as the detection algorithm would adapt to this behavior. IFTM models provide iterative learning and prediction for point-wise inputs. Therefore, all evaluated models meet the 6^{th} property of streaming analysis capabilities.

In Chapter 6, we introduced the hyperparameter optimization approach applicable to any IFTM model aiming to provide a solution for the 2^{nd} property. Due to its construction, individual hyperparameter configurations of models are automatically optimized. The hyperparameter optimization strategy has its own hyperparameters, but these are generalized and enable ML inexperienced users to apply a wide range of different anomaly detection techniques.

The rest of the requirements focus on the quantitative and qualitative performance of the operated detection approach. These provide further discrimination between the different IFTM approaches. We showed the overall runtime (see Figure 7.10), which captures the learning time as well as the prediction time for a single data point. The results illustrate that there exist roughly three levels of order in execution time. At first, there exist the Arima models and CABIRCH, which show near-zero execution time (less than 1ms). Secondly, AE and VAE, show higher execution times between 10ms and 30ms. Third, AELSTM and LSTM show the highest times with more than 30ms execution time, but still lower than 60ms. With an expected sampling rate of 500ms, all the given approaches provide just-in-time processing capabilities (5^{th} property). As the computation highly depends on the sampling rate, on the number of metrics monitored and on the hardware itself, applying of selected methods should be revised based on the defined three levels of the algorithm.

The 3^{rd} property was introduced to reflect the problems of supervised learning approaches when applying them on continuous data streams. But the 3^{rd} property helps to differentiate among the algorithms by reflecting the three levels of runtime and combining the meaning with respect to the sampling rate. The first level with the least execution time finishes its calculations faster than $\frac{1ms}{500ms} = 0.2\%$ of time compared to the sampling rate. The second level needs roughly between 2% and 6% of time, while the third level utilizes up to 12%. Such results can vary depending on the utilized hardware. As the above described hardware was used for these experiments (see Section 7.1), we expect that currently more powerful, modern hardware solutions are established and the relation consequently shows beneficial results. We suggest that all given approaches provide suitable results to be placed in several different hardware components. The computational overhead is approximated by ratio of execution time needed with respect to the sampling rate as this provides the key indicator of CPU usage compared to the rest of available time from the monitoring interval, where the CPU is free to execute the actual cloud computing component. The results show, that UArima, MArima, and CABIRCH provide the best ratio. Especially, when planning capacities when applying several component analysis on a single physical device. For example, public cloud providers can monitor their services running in VMs indirectly from the hypervisor layer (e.g. via libvirt). Thus, multiple IFTM models run in parallel to analyze the different VMs. Public cloud providers may provide VM images in the future, which collect further detailed information, enabling the analysis to provide white-box information and consequently the possibility to benefit from e.g. increased analysis accuracy. This holds also for private cloud providers, which are able to influence the software usage within virtual machines. With respect to IoT-gateway monitoring or edge clouds, software components can be analyzed but currently suffer from computational power compared to dedicated servers. The anomaly detection algorithms should consequently also cope with lower computational power. We expect that UArima, MArima as well as CABIRCH are able to provide beneficial computa-

tional properties for such devices. Utilizing small-sized neural networks on IoT-devices may also be applicable and we expect that in future the computational power grows and therefore larger models are going to be applicable to be deployed on such devices soon.

Besides the computational overhead, the 4^{th} property also describes the demand for high accuracy with aimed 99.999%. For point-wise evaluation (see Table C.1 and Table C.2 in Appendix C), all given approaches do not meet the aimed 99.999% accuracy. As the evaluation captures unsupervised approaches, this high demanded accuracy cannot be met for point-wise predictions due to the lack of labeled knowledge. On the other hand, the unsupervised approaches provide flexibility for the applicability as industrial providers do not specialize in labeling large amounts of data. Unsupervised approaches can cope with the lack of labeled training data and can therefore be applied. Figure 7.2 showed the overall results of the different IFTM models with respect to the event detection rates and the TN-rates. Both values are crucial to consider as machine learning models are able to either overfit (FN-rate increases) the problem or generalize too much (FP-rate increases). The best algorithm to consider provides a balance between those tendencies. Therefore, applicable anomaly detection approaches should be capable of detecting all given events and provide high TN-rates. The selection of an appropriate algorithm is dependent on the user's attitude to risk and whether the impact to the system is at higher risk for higher FN-rates or FP-rates.

For this work, we expect that at least all events have to be able to be detected. Building a model to predict each single data point to be abnormal is easy to be realized, it would provide high FP-rates and therefore should not be considered. The TN-rate is presented in Figure 7.2. Considering the top three approaches with respect to 100% detection rate and then sorted by the TN-rate, the user should consider the approaches AE & DEMM (98.27% TN-rate), LSTM & EMM (96.14% TN-rate) and CABIRCH & EMM (94.57% TN-rate).

Besides the demand of a high TN-rate and that all abnormal events should be detected, it is also important to consider when the event was detected. This requirement is captured by the 7^{th} property, which states that the anomaly should be detected as soon as possible with respect to the start of the anomaly event. The early detection gives the opportunity to provide more time to choose an appropriate remediation action in order to solve the anomaly. Furthermore, when there is a lot of time until a failure is expected, different actions can be considered e.g. by taking risk impact to the system into account.

Figure 7.7 illustrates the detection delay with respect to the TN-rate. The scatter plot shows that with decreasing TN-rate (unwanted behavior), the delay also decreases (wanted behavior). The user can select an appropriate detection algorithm based on its own preferences and needs based on the plot. The user should therefore consider on the one hand how much reaction time is needed to ensure low-risk impact of false alarms. On the other hand, the user might not expect high risks or is able to cope with higher false alarms by applying false alarm reduction mechanisms through e.g. intelligent analysis algorithms or human expert knowledge. In this case, one can benefit by selecting an anomaly detection algorithm with smaller TN-rate (e.g. $< 90\%$), but ensuring also a smaller detection delay ($< 10s$ respectively).

Including the above described runtime to the selection process, Figure 7.8 depicts the dependent information of those three key aspects. The best performing algorithms are described in the previous section, but we like to highlight the recommendation of selected algorithms next. Focusing on the detection rate and TN-rate, we recommend applying AE & DEMM as this IFTM model performs the lowest number of false alarms. In contrast, this model produced the highest detection delay, which still might provide applicable usage. Furthermore, the execution time is in the second level of order (as described above) compared to all other approaches and should be applicable to any modern server setup, but might be problematic when transferring this task to e.g.

IoT devices. When there exist enough computation resources and small requirements towards the detection delay, AE & DEMM should be recommended.

In case that high TN-rates as well as a smaller detection delay are important (compared to AE & DEMM), but the runtime is not required to provide best performance, the user should apply LSTM & EMM. But as the runtime and detection delay is much higher compared to CABIRCH & EMM and the difference in TN-rate is smaller than 2%, we recommend choosing the option CABIRCH & EMM when low detection times are assumed for higher-level analysis.

7.4 Future Work

Based on the given contribution towards a zero-touch anomaly detection solution for resource monitoring in cloud environments, there exist different aspects where this work contributes to new starting points to establish new research directions as well as a basis to include further aspects, which were out of scope in this thesis.

As the results showed, one of the key problems for unsupervised anomaly detection are false alarms in the normal signal. In order to increase the quality of results, there exist three options to focus on:

- The *preprocessing phase* provides the opportunity to apply feature engineering techniques changing the multivariate signal in that way to capture larger amount of discriminatory information. Consequently, the anomaly detection can distinguish more easily based on the incoming data stream. This process includes e.g. filtering of duplicated metrics with respect to the information carried, smoothing of the signal, noise reduction, feature construction.
- *White box* in-depth analysis of the individual algorithmic functionality and its behavior in the applied domain can be investigated in detail. Especially the combination between the preprocessing and the model characteristics are going to provide helpful insights about the possibility to model more specialized AI models for concrete services. This might be possible as e.g. databases and load balancers provide different normal behavior patterns.
- After receiving the results, *postprocessing analysis* steps can be applied like smoothing techniques, state analysis algorithms, and root cause analysis to foresee and avoid false alarms.

In the case of false alarm reduction, future work may concentrate also on the extension of the analytics pipeline conducted in the ZerOps design. For example, by introducing feedback mechanisms, the following analysis steps may provide further information to the anomaly detection algorithm like system load changes in order to provide adaptations to e.g. the hyperparameters to enable higher learning rates to adapt the AI model to the load change. In addition, the feedback loops provide the opportunity of whether anomalies were correctly detected to distinguish data points to be learned for the representation of the normal system state.

Generalization of the AI models can be investigated in future in order to develop a database of AI models for e.g. groups of similar models. When a new microservice instance is deployed, existing models can then be loaded and used without the need to provide hyperparameter tuning, mitigating the cold start problem even more. Through federated learning [283], generalized models for even different types of microservices might be developed in future.

The IFTM framework can be extended to provide unsupervised anomaly detection for additional types of monitored system data. This includes for example tracing data as well as log data. Combining these different types of data into a single AI model can improve the quality of results due to the additional information as well as providing

the chance to detect further types of anomalies, which are not recognizable by resource metrics.

Besides including further information sources to the algorithm, the IFTM model provides by design the opportunity to apply forecasting algorithms as identity functions. When utilizing e.g. Arima models to forecast not only the next value, but Arima can forecast further data points into the future. These resulting residuals can be investigated and analyzed on whether the data stream is going to be abnormal in the near future. This could provide additional information to the remediation engine about timing values until when degraded state anomalies might cause crash failures and provide hints about implications towards severity in order to select appropriate actions with respect to risk and time.

Lastly, IFTM can be applied to other domains of multivariate anomaly detection in data streams. We showed the applicability of IFTM models for morphological anomaly detection in medical ECG signals [284]. Besides other interdisciplinary domains, there exist within the computer science domain further opportunities to apply IFTM. For example, for IT-security threats like DDoS attacks can be also seen as a type of anomaly.

Chapter 8

Conclusion

The major aim of the thesis is the investigation, development and evaluation of the applicability of anomaly detection techniques to a holistic AIOps platform enhancing zero-touch administration for highly dependable IT-infrastructures. The objective concentrates on resource monitoring as a data source.

Chapter 4 described the holistic AIOps platform ZerOps, which provides a decentralized execution of monitoring agents and anomaly detection agents. The applicability and integration possibilities of supervised, semi-supervised and unsupervised machine learning techniques were discussed. In the case of supervised and semi-supervised learning, extensive evaluations on a cloud monitoring dataset were applied. The results showed the applicability of supervised approaches in order to detect the different types of system anomalies. The decision tree classifier $J48$ reached the highest score with more than 99% accuracy. The results show the potential for AI models to be applied in this domain, but supervised models suffer from unrealistic assumptions of the existence of labeled data as these have to be maintained by e.g. experts. Furthermore, supervised models assume to have knowledge about all possible anomalies as they differentiate to a finite number of classes.

In order to overcome this problem, we showed how semi-supervised one-class learning provides the opportunity to model labeled normal behavior. Any differentiation to the normal behavior is reported as an anomaly. The evaluation illustrated the applicability of semi-supervised approaches but also showed a drop in accuracy with respect to supervised learners, which was expected because of the lack of information about abnormal behavior. Semi-supervised learning still assumes labeled data of the normal system behavior, which is in general widely available, but still needs the selection and labeling by e.g. experts.

In Chapter 5, we propose CABIRCH to provide a valuable solution towards zero-touch administration. CABIRCH is an unsupervised anomaly detection approach based on the semi-supervised clusterer BIRCH. CABIRCH is capable of adapting its CF-tree to the current data stream through micro-cluster aging. Assuming that most of the signal is normal, CABIRCH continuously trains the signal and forgets older clusters over time. For anomaly detection, we employed CABIRCH into IFTM applying dynamic thresholding for distinguishing abnormal data. We showed the applicability of CABIRCH to detect anomalies and provided detailed behavior analysis to several different anomaly patterns. CABIRCH provides change point detection capabilities for rapid changes and fluctuation changes, while constant change detection for successive patterns when applied to constant learning. In the case of applying CABIRCH in a tumbling window setting, as described in Chapter 5, sequences of abnormal behavior can be detected for all of the anomaly patterns. This of course, is highly dependable on the tumbling window size as too short windows would provide adaption to long-lasting

anomaly patterns, while too large windows might lead to too slow concept adaption to the normal signal behavior.

Although CABIRCH is capable of unsupervised anomaly detection, hyperparameters have to be configured in order to provide accurate detection results. For a zero-touch solution, we introduce a hyperparameter optimization approach with a novel fitness function definition (see Chapter 6). The hyperparameter optimization applies genetic programming to infer suitable values to optimize the fitness function. The fitness score captures IFTM specific parameters, which do not require the need for labeled information. We showed that the fitness function correlates to the accuracy and we illustrated the convergence towards high fitness scores over time. Through this approach, we are able to provide a wide range of unsupervised anomaly detection algorithms hyperparameter optimization enabling non-expert usage of ML models to cloud monitoring.

Based on these findings, we conducted an evaluation of 28 different IFTM models including combinations of anomaly detection approaches from the related work. The evaluation results show the applicability to the domain with high detection rates of abnormal events and a low number of false alarms in the normal signal (high TN-rate). The top three identified approaches are: AE & DEMM (98.27% TN-rate), LSTM & EMM (96.14% TN-rate) and CABIRCH & EMM (94.57% TN-rate). The selection depends on the demand of the user's system requirements and further analysis algorithms contained in the self-healing pipeline, which were discussed in Chapter 7.

With respect to the research objectives (see Section 1.1): RO-1 defines upon the challenges and available strategies general requirements for ML-based anomaly detection in this application domain in Chapter 4. RO-2 aims to provide further guidance to narrow down applicable solutions meeting the requirements, which are also discussed in Chapter 4). Lastly, a solution is developed, implemented in Chapters 5-6 and evaluated in Chapter 7 towards a solution for zero-touch administration (RO-3).

Limitations When applying CABIRCH with continuous learning, it is capable of detecting change points, which recognize the beginning and the ending of e.g. abnormal behavior. As such change points are mostly referred to single data points, it is difficult to differentiate those to false alarms. We therefore provided the option of running CABIRCH in a tumbling window, but the tumbling window is highly dependable on the window size.

To overcome this limitation, our proposed hyperparameter optimization approach can be applied to determine suited window sizes. When applying the hyperparameter optimization, the technique suffers from the same problem as it also utilizes a tumbling window. Here, the problem arises that for long-lasting anomalies as the ML models are going to be adapted towards hyperparameters for covering this abnormal behavior best. This can be intentional, as long-lasting anomalies can be seen as chronic behavior, where remediation processes could not determine mitigating actions, but the anomaly stays in a degraded state without failure.

To overcome the cold start problem with fast converging optimal hyperparameter optimization, a wide range of different values should be applied in order to enhance the probability to find the global optimal values, resulting in a high population of ML models. Due to resource limitations, the population size is also limited. Consequently, local optima might be converged, which are far away from the global optimum. The genetic algorithm encounters this problem by applying mutations to find not yet seen values. Still, this process might be time demanding. On the other hand, the search space can be reduced by limiting the search intervals of the individual hyperparameters. As this requires domain-specific knowledge as well as knowledge about the specific ML model, automatic methods should be investigated. Crowd source approaches might help to build up databases of ML models for similar system configurations and services in the future.

Unsupervised approaches arrive with problematic drawbacks due to the lack of pre-labeled training data. The point-wise detection results show that such techniques suffer from higher false alarm rates. On the other hand, unsupervised techniques can be applied out of the box. In order to approach the problem of false alarms, we provided options for future work in Chapter 7.

In conclusion, this work showed the applicability of anomaly detection approaches for cloud monitoring and provides guidance for selecting appropriate detection approaches with respect to demanded requirements. Especially the aspects towards a zero-touch solution for a holistic AIOps platform were presented and discussed. The results of the anomaly detection will be a starting point for future higher-level analysis techniques providing holistic self-healing capabilities.

Bibliography

[1] Mark Hung. Leading the iot, gartner insights on how to lead in a connected world. *Gartner Research*, pages 1–29, 2017.

[2] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 1, 2018.

[3] Eric Bauer, Xuemei Zhang, and Douglas A Kimber. *Practical system reliability*. John Wiley & Sons, 2009.

[4] Mohammad Hossein Ghahramani, MengChu Zhou, and Chi Tin Hon. Toward cloud computing qos architecture: Analysis of cloud systems and cloud services. *IEEE/CAA Journal of Automatica Sinica*, 4(1):6–18, 2017.

[5] Jennifer Davis and Ryn Daniels. *Effective DevOps: building a culture of collaboration, affinity, and tooling at scale*. " O'Reilly Media, Inc.", 2016.

[6] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems*. " O'Reilly Media, Inc.", 2016.

[7] Inc. Gartner. AIOps Market Guide, 2019. Available online: https://www.bmc.com/blogs/gartner-aiops-market-guide/ (Accessed 30-10-2019).

[8] Anton Gulenko, Marcel Wallschläger, Florian Schmidt, Odej Kao, and Feng Liu. A system architecture for real-time anomaly detection in large-scale nfv systems. *Procedia Computer Science*, 94:491–496, 2016.

[9] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[10] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[11] Vic Barnett and Toby Lewis. *Outliers in statistical data*. Wiley, 1974.

[12] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[13] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22(2):85–126, 2004.

[14] Mia Hubert and Ellen Vandervieren. An adjusted boxplot for skewed distributions. *Computational statistics & data analysis*, 52(12):5186–5201, 2008.

[15] Charu C Aggarwal. High-dimensional outlier detection: The subspace method. In *Outlier Analysis*, pages 135–167. Springer, 2013.

[16] Leonid Kalinichenko, Ivan Shanin, and Ilia Taraban. Methods for anomaly detection: A survey. In *CEUR Workshop Proceedings*, 2014.

[17] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.

[18] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[19] Nan Jiang and Le Gruenwald. Research issues in data stream association rule mining. *ACM Sigmod Record*, 35(1):14–19, 2006.

[20] Shehroz S Khan and Michael G Madden. A survey of recent trends in one class classification. In *Irish conference on artificial intelligence and cognitive science*, pages 188–197. Springer, 2009.

[21] Markus Goldstein and Seiichi Uchida. Behavior analysis using unsupervised anomaly detection. In *The 10th Joint Workshop on Machine Perception and Robotics (MPR 2014). Online*, 2014.

[22] Anitha Ramchandran and Arun Kumar Sangaiah. Unsupervised anomaly detection for high dimensional data—an exploratory analysis. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, pages 233–251. Elsevier, 2018.

[23] Shiblee Sadik and Le Gruenwald. Research issues in outlier detection for data streams. *Acm Sigkdd Explorations Newsletter*, 15(1):33–40, 2014.

[24] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1):11–33, 2004.

[25] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204, 2010.

[26] Jim Gray. A census of tandem system availability between 1985 and 1990. *IEEE Transactions on reliability*, 39(4):409–418, 1990.

[27] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[28] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX symposium on internet technologies and systems*, volume 67. Seattle, WA, 2003.

[29] Domenico Cotroneo, Roberto Pietrantuono, Stefano Russo, and Kishor Trivedi. How do bugs surface? a comprehensive study on the characteristics of software bugs manifestation. *Journal of Systems and Software*, 113:27–43, 2016.

[30] Yennun Huang, Chandra Kintala, Nick Kolettis, and N Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *Twenty-fifth international symposium on fault-tolerant computing. Digest of papers*, pages 381–390. IEEE, 1995.

[31] Vittorio Castelli, Richard E Harper, Philip Heidelberger, Steven W Hunter, Kishor S Trivedi, Kalyanaraman Vaidyanathan, and William P Zeggert. Proactive management of software aging. *IBM Journal of Research and Development*, 45(2):311–332, 2001.

[32] Alberto Avritzer and Elaine J Weyuker. Monitoring smoothly degrading systems for increased dependability. *Empirical Software Engineering*, 2(1):59–77, 1997.

[33] Michael Grottke, Rivalino Matias, and Kishor S Trivedi. The fundamentals of software aging. In *2008 IEEE International conference on software reliability engineering workshops (ISSRE Wksp)*, pages 1–6. Ieee, 2008.

[34] Massimo Condoluci, Fragkiskos Sardis, and Toktam Mahmoodi. Softwarization and virtualization in 5g networks for smart cities. In *International Internet of Things Summit*, pages 179–186. Springer, 2015.

[35] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018.

[36] Clearwater Service Architecture, 2019. Available online: https://www.daitan.com/wp-content/uploads/2015/08/clearwater.png (Accessed 31-10-2019).

[37] CORD Design Notes. Central office re-architected as a datacenter (cord), 2016.

[38] Larry Peterson, Ali Al-Shabibi, Tom Anshutz, Scott Baker, Andy Bavier, Saurav Das, Jonathan Hart, Guru Palukar, and William Snow. Central office re-architected as a data center. *IEEE Communications Magazine*, 54(10):96–101, 2016.

[39] Anton Gulenko, Florian Schmidt, Alexander Acker, Marcel Wallschläger, Odej Kao, and Feng Liu. Detecting anomalous behavior of black-box services modeled with distance-based online clustering. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 912–915. IEEE, 2018.

[40] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.

[41] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[42] Susanne Albers. Online algorithms. In *Interactive Computation*, pages 143–164. Springer, 2006.

[43] Greg Nelson and Jeff Wright. Real time decision support: creating a flexible architecture for real time analytics. *DSSResources. COM*, 11:18, 2005.

[44] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[45] Max Landauer, Markus Wurzenberger, Florian Skopik, Giuseppe Settanni, and Peter Filzmoser. Time series analysis: unsupervised anomaly detection beyond outlier detection. In *International Conference on Information Security Practice and Experience*, pages 19–36. Springer, 2018.

[46] Shan Suthaharan. Machine learning models and algorithms for big data classification. *Integr. Ser. Inf. Syst*, 36:1–12, 2016.

[47] Hendrik Fichtenberger, Marc Gillé, Melanie Schmidt, Chris Schwiegelshohn, and Christian Sohler. Bico: Birch meets coresets for k-means clustering. In *European Symposium on Algorithms*, pages 481–492. Springer, 2013.

[48] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC press, 2013.

[49] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 328–339. SIAM, 2006.

[50] Irene Ntoutsi, Arthur Zimek, Themis Palpanas, Peer Kröger, and Hans-Peter Kriegel. Density-based projected clustering over high dimensional data streams. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 987–998. SIAM, 2012.

[51] Tian Zhang. Data clustering for very large datasets plus applications. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1997.

[52] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.

[53] D. T. Grozdic and S. T. Jovicic. Whispered speech recognition using deep denoising autoencoder and inverse filtering. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(12):2313–2322, Dec 2017.

[54] Kun Zeng, Jun Yu, Ruxin Wang, Cuihua Li, and Dacheng Tao. Coupled deep autoencoder for single image super-resolution. *IEEE transactions on cybernetics*, 47(1):27–37, 2017.

[55] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[56] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[57] P Kingma Diederik, Max Welling, et al. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[58] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[59] LR Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5, 2001.

[60] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[61] David MQ Nelson, Adriano CM Pereira, and Renato A de Oliveira. Stock market's price movement prediction with lstm neural networks. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1419–1426. IEEE, 2017.

[62] Jung-Woo Ha, Adrian Kim, Dongwon Kim, Jeonghee Kim, Jeong-Whun Kim, Jin Joo Park, and Borim Ryu. Predicting high-risk prognosis from diagnostic histories of adult disease patients via deep recurrent neural networks. In *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, pages 394–399. IEEE, 2017.

[63] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4694–4702. IEEE, 2015.

[64] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[65] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395. ACM, 2018.

[66] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[67] Dong Oh and Il Yun. Residual error based anomaly detection using auto-encoder in smd machine sound. *Sensors*, 18(5):1308, 2018.

[68] Valeriy Zakamulin. *Market Timing with Moving Averages: The Anatomy and Performance of Trading Rules*. Springer, 2017.

[69] J Stuart Hunter. The exponentially weighted moving average. *Journal of quality technology*, 18(4):203–210, 1986.

[70] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[71] Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

[72] Jinn-Tsong Tsai, Jyh-Horng Chou, and Tung-Kuan Liu. Tuning the structure and parameters of a neural network by using hybrid taguchi-genetic algorithm. *IEEE Transactions on Neural Networks*, 17(1):69–80, 2006.

[73] Stefan Lessmann, Robert Stahlbock, and Sven F Crone. Optimizing hyperparameters of support vector machines by genetic algorithms. In *IC-AI*, pages 74–82, 2005.

[74] Steven R Young, Derek C Rose, Thomas P Karnowski, Seung-Hwan Lim, and Robert M Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 4. ACM, 2015.

[75] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 497–504. ACM, 2017.

[76] Stefano Cagnoni, Andrew B Dobrzeniecki, Riccardo Poli, and Jacquelyn C Yanch. Genetic algorithm-based interactive segmentation of 3d medical images. *Image and Vision Computing*, 17(12):881–895, 1999.

[77] Markus Gudmundsson, Essam A El-Kwae, and Mansur R Kabuka. Edge detection in medical images using a genetic algorithm. *IEEE transactions on medical imaging*, 17(3):469–474, 1998.

[78] Payel Ghosh and Melanie Mitchell. Segmentation of medical images using a genetic algorithm. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1171–1178. ACM, 2006.

[79] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2):125–132, 2000.

[80] Franklin Allen and Risto Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of financial Economics*, 51(2):245–271, 1999.

[81] Ping-Feng Pai and Wei-Chiang Hong. Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms. *Electric Power Systems Research*, 74(3):417–425, 2005.

[82] Mohsen Nasseri, Keyvan Asghari, and MJ Abedini. Optimized scenario for rainfall forecasting using genetic algorithm coupled with artificial neural network. *Expert systems with applications*, 35(3):1415–1421, 2008.

[83] A Sedki, Driss Ouazar, and E El Mazoudi. Evolving neural network using real coded genetic algorithm for daily rainfall–runoff forecasting. *Expert Systems with Applications*, 36(3):4523–4527, 2009.

[84] Ting Yee Lim. Structured population genetic algorithms: a literature survey. *Artificial Intelligence Review*, 41(3):385–399, 2014.

[85] JD Da Silva and PO Simoni. The island model parallel ga and uncertainty reasoning in the correspondence problem. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 2247–2252. IEEE, 2001.

[86] Enrique Alba, Hugo Alfonso, and Bernabé Dorronsoro. Advanced models of cellular genetic algorithms evaluated on sat. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1123–1130. ACM, 2005.

[87] Gregory S Hornby. Alps: the age-layered population structure for reducing the problem of premature convergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 815–822. ACM, 2006.

[88] Reza Akbari, Vahid Zeighami, and Koorush Ziarati. Mlga: A multilevel cooperative genetic algorithm. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 271–277. IEEE, 2010.

[89] JB Brown. Classifiers and their metrics quantified. *Molecular informatics*, 37(1-2):1700127, 2018.

[90] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[91] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[92] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. Bug characteristics in open source software. *Empirical software engineering*, 19(6):1665–1705, 2014.

[93] Jim Gray. Why do computers stop and what can be done about it? In *Symposium on reliability in distributed software and database systems*, pages 3–12. Los Angeles, CA, USA, 1986.

[94] George Candea, Stefan Bucur, and Cristian Zamfir. Automated software testing as a service. In *Proceedings of ACM Symposium on Cloud Computing*, pages 155–160. ACM, 2010.

[95] Michael Pradel and Thomas R Gross. Leveraging test generation and specification mining for automated bug detection without false positives. In *Proceedings of Intl. Conference on Software Engineering*, pages 288–298. IEEE Press, 2012.

[96] Michael Grottke and Kishor S Trivedi. Fighting bugs: Remove, retry, replicate, and rejuvenate. *Computer*, 40(2):107–109, 2007.

[97] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1):1–34, 2014.

[98] Sachin Garg, Aad Van Moorsel, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. A methodology for detection and estimation of software aging. In *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No. 98TB100257)*, pages 283–292. IEEE, 1998.

[99] Jean Araujo, Rubens Matos, Paulo Maciel, Rivalino Matias, and Ibrahim Beicker. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In *Proceedings of the Middleware 2011 Industry Track Workshop*, pages 1–7, 2011.

[100] Carlos Melo, Jean Araujo, Vandi Alves, and Paulo Romero Martins Maciel. Investigation of software aging effects on the openstack cloud computing platform. *JSW*, 12(2):125–137, 2017.

[101] João Paulo Magalhães and Luis Moura Silva. Prediction of performance anomalies in web-applications based-on software aging scenarios. In *2010 IEEE second international workshop on software aging and rejuvenation*, pages 1–7. IEEE, 2010.

[102] Hai-Ning Meng, Yong Qi, Di Hou, and Ying Chen. A rough wavelet network model with genetic algorithm and its application to aging forecasting of application server. In *2007 International Conference on Machine Learning and Cybernetics*, volume 5, pages 3034–3039. IEEE, 2007.

[103] Karen J Cassidy, Kenny C Gross, and Amir Malekpour. Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. In *Proceedings international conference on dependable systems and networks*, pages 478–482. IEEE, 2002.

[104] Michael Grottke, Lei Li, Kalyanaraman Vaidyanathan, and Kishor S Trivedi. Analysis of software aging in a web server. *IEEE Transactions on reliability*, 55(3):411–420, 2006.

[105] Jun Zhang, Yang Xiang, Yu Wang, Wanlei Zhou, Yong Xiang, and Yong Guan. Network traffic classification using correlation information. *IEEE Transactions on Parallel and Distributed systems*, 24(1):104–117, 2012.

[106] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, 2017.

[107] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[108] J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[109] Shailendra Sahu and Babu M Mehtre. Network intrusion detection system using j48 decision tree. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2023–2026. IEEE, 2015.

[110] Shadi Aljawarneh, Muneer Bani Yassein, and Mohammed Aljundi. An enhanced j48 classification algorithm for the anomaly intrusion detection systems. *Cluster Computing*, 22(5):10549–10565, 2019.

[111] R Delshi Howsalya Devi and P Deepika. An automated diagnosis of breast cancer using farthest first clustering and decision tree j48 classifier. *Advances in Natural and Applied Sciences*, 10(10 SE):161–167, 2016.

[112] John Richard D Kho and Larry A Vea. Credit card fraud detection based on transaction behavior. In *TENCON 2017-2017 IEEE Region 10 Conference*, pages 1880–884. IEEE, 2017.

[113] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine learning*, 59(1-2):161–205, 2005.

[114] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[115] Bayu Adhi Tama and Kyung-Hyune Rhee. Hfste: Hybrid feature selections and tree-based classifiers ensemble for intrusion detection system. *IEICE TRANSACTIONS on Information and Systems*, 100(8):1729–1737, 2017.

[116] Kyriakos Stefanidis and Artemios G Voyiatzis. An hmm-based anomaly detection approach for scada systems. In *IFIP International Conference on Information Security Theory and Practice*, pages 85–99. Springer, 2016.

[117] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM, 2001.

[118] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[119] Kajal Rai, M Syamala Devi, and Ajay Guleria. Decision tree based algorithm for intrusion detection. *International Journal of Advanced Networking and Applications*, 7(4):2828, 2016.

[120] Mrudula Gudadhe, Prakash Prasad, and Lecturer Kapil Wankhade. A new data mining based network intrusion detection model. In *2010 International Conference on Computer and Communication Technology (ICCCT)*, pages 731–735. IEEE, 2010.

[121] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[122] Asmah Muallem, Sachhin Shetty, Jan W Pan, Juan Zhao, and Biswajit Biswal. Hoeffding tree algorithms for anomaly detection in streaming datasets: A survey. *Journal of Information Security*, 8(4), 2017.

[123] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[124] Yaqi Li, Chun Yan, Wei Liu, and Maozhen Li. Research and application of random forest model in mining automobile insurance fraud. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1756–1761. IEEE, 2016.

[125] Charissa Ann Ronao and Sung-Bae Cho. Anomalous query access detection in rbac-administered databases with random forest and pca. *Information Sciences*, 369:238–250, 2016.

[126] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[127] Abdallah Abbey Sebyala, Temitope Olukemi, Lionel Sacks, and Dr Lionel Sacks. Active platform security through intrusion detection using naive bayesian network for anomaly detection. In *London Communications Symposium*, pages 1–5. Citeseer, 2002.

[128] Z Muda, W Yassin, MN Sulaiman, and NI Udzir. Intrusion detection based on k-means clustering and naïve bayes classification. In *2011 7th International Conference on Information Technology in Asia*, pages 1–6. IEEE, 2011.

[129] Shweta Kharya, Shika Agrawal, and Sunita Soni. Naive bayes classifiers: A probabilistic detection model for breast cancer. *International Journal of Computer Applications*, 92(10):0975–8887, 2014.

[130] George Dimitoglou, James A Adams, and Carol M Jim. Comparison of the c4. 5 and a naïve bayes classifier for the prediction of lung cancer survivability. *arXiv preprint arXiv:1206.1121*, 2012.

[131] W Nor Haizan W Mohamed, Mohd Najib Mohd Salleh, and Abdul Halim Omar. A comparative study of reduced error pruning method in decision tree algorithms. In *2012 IEEE International conference on control system, computing and engineering*, pages 392–397. IEEE, 2012.

[132] Wayne Iba and Pat Langley. Induction of one-level decision trees. In *Machine Learning Proceedings 1992*, pages 233–240. Elsevier, 1992.

[133] Ron Kohavi. The power of decision tables. In *European conference on machine learning*, pages 174–189. Springer, 1995.

[134] William W Cohen. Fast effective rule induction. In *Machine learning proceedings 1995*, pages 115–123. Elsevier, 1995.

[135] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[136] Eibe Frank and Ian H Witten. Generating accurate rule sets without global optimization. 1998.

[137] John C Platt. Fast training of support vector machines using sequential minimal optimization, advances in kernel methods. *Support Vector Learning*, pages 185–208, 1999.

[138] S Saibharath and G Geethakumari. Cloud forensics: evidence collection and preliminary analysis. In *2015 IEEE International Advance Computing Conference (IACC)*, pages 464–467. IEEE, 2015.

[139] Neeraj Chavan, Fabio Di Troia, and Mark Stamp. A comparative analysis of android malware. *arXiv preprint arXiv:1904.00735*, 2019.

[140] PSS Siva Krishna and P Venkateswara Rao. An analysis of malware classification technique by using machine learning. *International Journal of Computer Applications*, 975:8887.

[141] Mohsen Rezvani. Assessment methodology for anomaly-based intrusion detection in cloud computing. *Journal of AI and Data Mining*, 6(2):387–397, 2018.

[142] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2):18–28, 2009.

[143] Carla Sauvanaud, Kahina Lazri, Mohamed Kaâniche, and Karama Kanoun. Anomaly detection and root cause localization in virtual network functions. In *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*, pages 196–206. IEEE, 2016.

[144] Kirila Adamova, Dominik Schatzmann, Bernhard Plattner, and Paul Smith. Network anomaly detection in the cloud: The challenges of virtual service migration. In *2014 IEEE International Conference on Communications (ICC)*, pages 3770–3775. IEEE, 2014.

[145] Tian Huang, Yongxin Zhu, Yafei Wu, Stéphane Bressan, and Gillian Dobbie. Anomaly detection and identification scheme for vm live migration in cloud infrastructure. *Future Generation Computer Systems*, 56:736–745, 2016.

[146] Zakia Ferdousi and Akira Maeda. Unsupervised outlier detection in time series data. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pages x121–x121. IEEE, 2006.

[147] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1939–1947. ACM, 2015.

[148] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261. ACM, 2003.

[149] Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. Statistical techniques for online anomaly detection in data centers. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 385–392. IEEE, 2011.

[150] Jeffrey P Buzen and Annie W Shum. Masf-multivariate adaptive statistical filtering. In *Int. CMG Conference*, pages 1–10, 1995.

[151] Maciej Szmit and Anna Szmit. Usage of modified holt-winters method in the anomaly detection of network traffic: Case studies. *Journal of Computer Networks and Communications*, 2012, 2012.

[152] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Dan Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. Argus: End-to-end service anomaly detection and localization from an isp's point of view. In *2012 Proceedings IEEE INFOCOM*, pages 2756–2760. IEEE, 2012.

[153] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM computer communication review*, volume 34, pages 219–230. ACM, 2004.

[154] Wei Xiao, Xiaolin Huang, Fan He, Jorge Silva, Saba Emrani, and Arin Chaudhuri. Online robust principal component analysis with change point detection. *IEEE Transactions on Multimedia*, 22(1):59–68, 2019.

[155] Mostafa Rahmani and Ping Li. Outlier detection and robust pca using a convex measure of innovation. In *Advances in Neural Information Processing Systems*, pages 14200–14210, 2019.

[156] Yu Gu, Andrew McCallum, and Don Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 32–32. USENIX Association, 2005.

[157] JBD Caberera, B Ravichandran, and Raman K Mehra. Statistical traffic modeling for network intrusion detection. In *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No. PR00728)*, pages 466–473. IEEE, 2000.

[158] Bernhard Schölkopf, John C Platt, John Shawe-Taylor, Alex J Smola, and Robert C Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

[159] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[160] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[161] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[162] Aarthi Reddy, Meredith Ordway-West, Melissa Lee, Matt Dugan, Joshua Whitney, Ronen Kahana, Brad Ford, Johan Muedsam, Austin Henslee, and Max Rao. Using gaussian mixture models to detect outliers in seasonal univariate network traffic. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 229–234. IEEE, 2017.

[163] M Bahrololum and M Khaleghi. Anomaly intrusion detection system using gaussian mixture model. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, volume 1, pages 1162–1167. IEEE, 2008.

[164] Rui Liu, Xiao-long Qian, Shu Mao, and Shuai-zheng Zhu. Research on anti-money laundering based on core decision tree algorithm. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 4322–4325. IEEE, 2011.

[165] Santosh Kumar, Sumit Kumar, and Sukumar Nandi. Multi-density clustering algorithm for anomaly detection using kdd'99 dataset. In *International Conference on Advances in Computing and Communications*, pages 619–630. Springer, 2011.

[166] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[167] Apapan Pumsirirat and Liu Yan. Credit card fraud detection using deep learning based on auto-encoder and restricted boltzmann machine. *International Journal of advanced computer science and applications*, 9(1):18–25, 2018.

[168] Alessandra De Paola, Salvatore Favaloro, Salvatore Gaglio, Giuseppe Lo Re, and Marco Morana. Malware detection through low-level features and stacked denoising autoencoders. In *ITASEC*, 2018.

[169] Tie Luo and Sai G Nagarajan. Distributed anomaly detection using autoencoder neural networks in wsn for iot. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[170] Shen Zhang, Fei Ye, Bingnan Wang, and Thomas G Habetler. Semi-supervised learning of bearing anomaly detection via deep variational autoencoders. *arXiv preprint arXiv:1912.01096*, 2019.

[171] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139. IEEE, 2016.

[172] Sucheta Chauhan and Lovekesh Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7. IEEE, 2015.

[173] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

[174] Jun Liu, Shuyu Chen, Zhen Zhou, and Tianshu Wu. An anomaly detection algorithm of cloud platform based on self-organizing maps. *Mathematical Problems in Engineering*, 2016, 2016.

[175] Jont B Allen and Lawrence R Rabiner. A unified approach to short-time fourier analysis and synthesis. *Proceedings of the IEEE*, 65(11):1558–1564, 1977.

[176] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196. International World Wide Web Conferences Steering Committee, 2018.

[177] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1), 2015.

[178] Olumuyiwa Ibidunmoye, Ali-Reza Rezaie, and Erik Elmroth. Adaptive anomaly detection in performance metric streams. *IEEE Transactions on Network and Service Management*, 15(1):217–231, 2017.

[179] Charu C Aggarwal. Outlier analysis. In *Data mining*, pages 237–263. Springer, 2015.

[180] Douglas C Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, 2007.

[181] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[182] Subutai Ahmad and Jeff Hawkins. Properties of sparse distributed representations and their application to hierarchical temporal memory. *arXiv preprint arXiv:1503.07469*, 2015.

[183] Domenico Cotroneo, Roberto Natella, and Stefano Rosiello. A fault correlation approach to detect performance anomalies in virtual network function chains. In *Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on*, pages 90–100. IEEE, 2017.

[184] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th international conference on Autonomic computing*, pages 191–200. ACM, 2012.

[185] Paul S Bradley, Usama M Fayyad, Cory Reina, et al. Scaling clustering algorithms to large databases. In *KDD*, volume 98, pages 9–15, 1998.

[186] Fredrik Farnstrom, James Lewis, and Charles Elkan. Scalability for clustering algorithms revisited. *SIGKDD explorations*, 2(1):51–57, 2000.

[187] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–4, 2012.

[188] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[189] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

[190] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems*, 29(2):249–272, 2011.

[191] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2):181–214, 2008.

[192] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.

[193] Pedro Pereira Rodrigues, João Gama, and Joao Pedroso. Hierarchical clustering of time-series data streams. *IEEE transactions on knowledge and data engineering*, 20(5):615–627, 2008.

[194] Joao Gama, Pedro Pereira Rodrigues, and Luís Lopes. Clustering distributed sensor data streams using local processing and reduced communication. *Intelligent Data Analysis*, 15(1):3–28, 2011.

[195] Boris Lorbeer, Ana Kosareva, Bersant Deva, Dženan Softić, Peter Ruppel, and Axel Küpper. Variations on the clustering algorithm birch. *Big data research*, 11:44–53, 2018.

[196] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[197] Dongwei Guo, Jingwen Chen, Yingjie Chen, and Zhiyu Li. Lbirch: An improved birch algorithm based on link. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pages 74–78. ACM, 2018.

[198] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. *Information systems*, 25(5):345–366, 2000.

[199] Yangyang Li, Guangyuan Liu, Peidao Li, and Licheng Jiao. A large-scale data clustering algorithm based on birch and artificial immune network. In *International Conference on Swarm Intelligence*, pages 327–337. Springer, 2018.

[200] Leandro Nunes de Castro and Fernando J Von Zuben. ainet: an artificial immune network for data analysis. In *Data mining: a heuristic approach*, pages 231–260. IGI Global, 2002.

[201] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[202] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[203] Samuel Xavier-de Souza, Johan AK Suykens, Joos Vandewalle, and Désiré Bollé. Coupled simulated annealing. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(2):320–335, 2009.

[204] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[205] Jake Graser, Steven K Kauwe, and Taylor D Sparks. Machine learning and energy minimization approaches for crystal structure predictions: A review and new horizons. *Chemistry of Materials*, 30(11):3601–3612, 2018.

[206] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, 65:83–92, 2016.

[207] Irene Samora, Mário J Franca, Anton J Schleiss, and Helena M Ramos. Simulated annealing in optimization of energy production in a water supply network. *Water resources management*, 30(4):1533–1547, 2016.

[208] L.M. Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. Simulated annealing algorithm for deep learning. *Procedia Computer Science*, 72:137 – 144, 2015. The Third Information Systems International Conference 2015.

[209] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[210] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[211] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[212] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[213] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[214] Oded Maron and Andrew W Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in neural information processing systems*, pages 59–66, 1994.

[215] D Anderson and K Burnham. Model selection and multi-model inference. *Second. NY: Springer-Verlag*, 63, 2004.

[216] WJ Conover. Practical nonparametric statistics, john wiley & sons. *INC, New York*, 1999.

[217] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43 – 58, 2016.

[218] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43. Ieee, 1995.

[219] Clara Marina Martínez and Dongpu Cao. 2 - integrated energy management for electrified vehicles. In Clara Marina Martínez and Dongpu Cao, editors, *Ihorizon-Enabled Energy Management for Electrified Vehicles*, pages 15 – 75. Butterworth-Heinemann, 2019.

[220] Peter Wilson and H. Alan Mantooth. Chapter 10 - model-based optimization techniques. In Peter Wilson and H. Alan Mantooth, editors, *Model-Based Engineering for Complex Electronic Systems*, pages 347 – 367. Newnes, Oxford, 2013.

[221] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950 Vol. 3, July 1999.

[222] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[223] Riccardo Poli. An analysis of publications on particle swarm optimization applications. *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[224] Wei Chen, Run-tong Zhang, Yong-ming Cai, and Fa-sheng Xu. Particle swarm optimization for constrained portfolio selection problems. In *2006 International Conference on Machine Learning and Cybernetics*, pages 2425–2429. IEEE, 2006.

[225] Fasheng Xu and Wei Chen. Stochastic portfolio selection based on velocity limited particle swarm optimization. In *2006 6th World Congress on Intelligent Control and Automation*, volume 1, pages 3599–3603. IEEE, 2006.

[226] Zhancheng Wang, Bufu Huang, Weimin Li, and Yangsheng Xu. Particle swarm optimization for operational parameters of series hybrid electric vehicle. In *2006 IEEE International Conference on Robotics and Biomimetics*, pages 682–688. IEEE, 2006.

[227] AS Elwer, SA Wahsh, MO Khalil, and AM Nur-Eldeen. Intelligent fuzzy controller using particle swarm optimization for control of permanent magnet synchronous motor for electric vehicle. In *IECON'03. 29th Annual Conference of the IEEE Industrial Electronics Society (IEEE Cat. No. 03CH37468)*, volume 2, pages 1762–1766. IEEE, 2003.

[228] Rui Xu, Georgios C Anagnostopoulos, and Donald C Wunsch. Multiclass cancer classification using semisupervised ellipsoid artmap and particle swarm optimization with gene expression data. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):65–77, 2007.

[229] David E Goldberg and John H Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[230] B. Sahmadi and D. Boughaci. Hybrid genetic algorithm with svm for medical data classification. In *2018 International Conference on Applied Smart Systems (ICASS)*, pages 1–6, Nov 2018.

[231] Jean-François Connolly, Eric Granger, and Robert Sabourin. Evolution of heterogeneous ensembles through dynamic particle swarm optimization for video-based face recognition. *Pattern Recognition*, 45(7):2460–2477, 2012.

[232] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC bioinformatics*, 7(1):125, 2006.

[233] Pablo Ribalta Lorenzo, Jakub Nalepa, Michal Kawulok, Luciano Sanchez Ramos, and José Ranilla Pastor. Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the genetic and evolutionary computation conference*, pages 481–488. ACM, 2017.

[234] Shih-Wei Lin, Kuo-Ching Ying, Shih-Chieh Chen, and Zne-Jung Lee. Particle swarm optimization for parameter determination and feature selection of support vector machines. *Expert systems with applications*, 35(4):1817–1824, 2008.

[235] Matteo Fischetti and Matteo Stringher. Embedded hyper-parameter tuning by simulated annealing. *arXiv preprint arXiv:1906.01504*, 2019.

[236] Yağız Nalçakan and Tolga Ensari. Decision of neural networks hyperparameters with a population-based algorithm. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 276–281. Springer, 2018.

[237] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[238] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[239] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[240] P. Kiss, D. Fonyó, and T. Horváth. Blaboo: A lightweight black box optimizer framework. In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 213–218, Aug 2018.

[241] Carmelo JA Bastos Filho, Fernando B de Lima Neto, Anthony JCC Lins, Antonio IS Nascimento, and Marilia P Lima. A novel search algorithm based on fish school behavior. In *2008 IEEE International Conference on Systems, Man and Cybernetics*, pages 2646–2651. IEEE, 2008.

[242] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[243] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

[244] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[245] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. ACM, 2019.

[246] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[247] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[248] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[249] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka: Automatic model selection and hyperparameter optimization in. *Anteil EPB*, page 81, 2019.

[250] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: Efficient and robust automated machine learning. In *Automated Machine Learning*, pages 113–134. Springer, 2019.

[251] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Automated Machine Learning*, pages 151–160. Springer, 2019.

[252] Pieter Gijsbers, Erin LeDell, Sébastien Poirier, Janek Thomas, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. In *6th ICML Workshop on Automated Machine Learning*, 2019.

[253] Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. Autone: Hyperparameter optimization for massive network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[254] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[255] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[256] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[257] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[258] Tomáš Pevnỳ. Loda: Lightweight on-line detector of anomalies. *Machine Learning*, 102(2):275–304, 2016.

[259] Bitflow collector, 2019. Available online: https://github.com/bitflow-stream/go-bitflow-collector/ (Accessed 31-10-2019).

[260] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[261] G a Padmanabhan and Prem Vrat. An eoq model for items with stock dependent consumption rate and exponential decay. *Engineering Costs and Production Economics*, 18(3):241–246, 1990.

[262] Christopher C Moser, Jonathan M Keske, Kurt Warncke, Ramy S Farid, and P Leslie Dutton. Nature of biological electron transfer. *Nature*, 355(6363):796, 1992.

[263] MH Zwietering, Il Jongenburger, FM Rombouts, and K Van't Riet. Modeling of the bacterial growth curve. *Appl. Environ. Microbiol.*, 56(6):1875–1881, 1990.

[264] Stasys Girdzijauskas and Dalia Štreimikiene. Application of logistic models for stock market bubbles analysis. *Journal of Business Economics and Management*, 10(1):45–51, 2009.

[265] Daniel Pauly. *Fish population dynamics in tropical waters: a manual for use with programmable calculators*, volume 8. WorldFish, 1984.

[266] FJ Richards. A flexible growth function for empirical use. *Journal of experimental Botany*, 10(2):290–301, 1959.

[267] Florian Schmidt, Anton Gulenko, Marcel Wallschläger, Alexander Acker, Vincent Hennig, Feng Liu, and Odej Kao. Iftm-unsupervised anomaly detection for virtualized network function services. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 187–194. IEEE, 2018.

[268] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschläger, Alexander Acker, and Odej Kao. Unsupervised anomaly event detection for cloud monitoring using online arima. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 71–76. IEEE, 2018.

[269] Florian Schmidt, Florian Suri-Payer, Anton Gulenko, Marcel Wallschläger, Alexander Acker, and Odej Kao. Unsupervised anomaly event detection for vnf service monitoring using multivariate online arima. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 278–283. IEEE, 2018.

[270] Ralf Tönjes, P Barnaghi, M Ali, A Mileo, M Hauswirth, F Ganz, S Ganea, B Kjærgaard, D Kuemper, Septimiu Nechifor, et al. Real time iot stream processing and large-scale data analytics for smart city applications. In *poster session, European Conference on Networks and Communications*. sn, 2014.

[271] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *International Semantic Web Conference*, pages 374–389. Springer, 2015.

[272] Steven CH Hoi and Rong Jin. Semi-supervised ensemble ranking. 2008.

[273] Furu Wei, Wenjie Li, and Shixia Liu. irank: A rank-learn-combine framework for unsupervised ensemble ranking. *Journal of the American Society for Information Science and Technology*, 61(6):1232–1243, 2010.

[274] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[275] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[276] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[277] Osbert Bastani, Yani Ioannou, Leonidas Lampropoulos, Dimitrios Vytiniotis, Aditya Nori, and Antonio Criminisi. Measuring neural net robustness with constraints. In *Advances in neural information processing systems*, pages 2613–2621, 2016.

[278] Valentina Timčenko and Slavko Gajin. Ensemble classifiers for supervised anomaly based network intrusion detection. In *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 13–19. IEEE, 2017.

[279] Juan Vanerio and Pedro Casas. Ensemble-learning approaches for network security and anomaly detection. In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 1–6. ACM, 2017.

[280] Saeed Haddadi Makhsous, Anton Gulenko, Odej Kao, and Feng Liu. High available deployment of cloud-based virtualized network functions. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*, pages 468–475. IEEE, 2016.

[281] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster, and André Brinkmann. Non-intrusive virtualization management using libvirt. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 574–579. IEEE, 2010.

[282] Ben Pfaff and Bruce Davie. The open vswitch database management protocol. 2013.

[283] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[284] Alexander Acker, Florian Schmidt, Anton Gulenko, Reinhard Kietzmann, and Odej Kao. Patient-individual morphological anomaly detection in multi-lead electrocardiography data streams. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 3841–3846. IEEE, 2017.

[285] Paolo Burlando, Renzo Rosso, Luis G Cadavid, and Jose D Salas. Forecasting of short-term rainfall using arma models. *Journal of Hydrology*, 144(1-4):193–211, 1993.

[286] Patrik Gustavsson and Jonas Nordström. The impact of seasonal unit roots and vector arma modelling on forecasting monthly tourism flows. *Tourism Economics*, 7(2):117–133, 2001.

[287] Wim van Drongelen. *Signal Processing for Neuroscientists: An Introduction to the Analysis of Physiological Signals*. Elsevier, 2006.

[288] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton University Press, 1994.

[289] Edward James Hannan. *Multiple time series*, volume 38. John Wiley & Sons, 2009.

[290] Chenghao Liu, Steven C. H. Hoi, Peilin Zhao, and Jianling Sun. Online arima algorithms for time series prediction. In *Proceedings of AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1867–1873. AAAI Press, 2016.

[291] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[292] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.

[293] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127, 1950.

[294] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Appendices

Appendix A

Online Arima

The background and variations applied to UArima and MArima are described in the following and were published by us in [268, 269].

Univariate Online Arima

Forecasting on time series is applied for many years [285–287] by using the Autoregressive Moving Average Model (ARMA) [288, 289] and focus on univariate time series forecasting. The ARMA model can identify latent correlation within the time series, allowing forecasting future data points. Based on ARMA, the model was modified to use a differentiation to capture more fine granular changes than linear dependencies, called ARIMA [288, 289]. But both ARMA and ARIMA assume that we know the complete time series in advance, so that there is no missing data. This is problematic when predicting on a potentially endless data stream.

To overcome this limitation, Liu et al. [290] proposed a game-theoretic framework for online learning to forecast values, called online ARIMA model. In their setting, an online player sequentially commits to a decision by forecasting the next value and consequently suffers a loss. Liu et al. assume that coefficient vectors are set by an adversary and are not disclosed to the learner at any time. Since the learning part has no ability to infer the actual noise term ε_t of a time step t, this is generated by the adversary, while remaining undisclosed to the learner. The learner predicts a value $\tilde{x_t}$ and learns the monitored value x_t, subsequently suffering a loss l_t at time t:

$$l_t(x_t, \tilde{x_t}) = l_t(x_t, \nabla^d \tilde{x_t} + \sum_{i=0}^{d-1} \nabla^i x_{t-1}) \tag{A.1}$$

Liu et al. [290] showed that the original model ARIMA(k, d, q) can be approximated by the model ARIMA($k + m$, d, 0). The noise term is dropped and is compensated by extending the window size of k to regress upon by m. A new $(k + m)$-dimensional coefficient vector γ weights past observations, formulating the model:

$$\tilde{x_t}(\gamma) = \sum_{i=1}^{k+m} \gamma_i \nabla^d x_{t-i} + \sum_{i=0}^{d-1} \nabla^i x_{t-1} \tag{A.2}$$

and subsequently the loss:

$$l_t(x_t, \tilde{x_t}) = l_t(x_t, \sum_{i=1}^{k+m} \gamma_i \nabla^d x_{t-i} + \sum_{i=0}^{d-1} \nabla^i x_{t-1}) \tag{A.3}$$

Furthermore, Liu et al. [290] described two suitable online convex optimization solvers to iteratively learn γ. The Online Newton Step (ONS) procedure [291] and Online Gradient Descent (ODG) [292] are discussed and both implemented within the IFTM framework. While the ODG variant is computationally more efficient, the ONS approach provides a more favorable regret boundary that entails more accurate predictions. Moreover, choosing a suitable learning rate is harder in the gradient descent setting, yet crucial to achieve a good model approximation. Nevertheless, the ODG approach might be favorable when the lag-window is large. Thus, we consider in the following the ONS model to be used.

In contrast to Liu et al. [290], we denote no discrete feature space for the coefficients and assume them to be continuous. We therefore randomly initialize $\gamma_i \in [-0.5, 0.5],\ \forall i \in [1, \ldots, w]$ and update them ensuing each observation. Further adaption made in the implementation for the IFTM framework compared to Liu et al. [290] proposal, within ONS, the inversion of the pseudo-Hessian Matrix A has to be computed, which is computationally expensive. Thus, we utilize the Shermann Morrison formula [293], which leads to more efficient computation of the inverse.

As ARIMA is usually investigated for univariate time series forecasting, we employ for an incoming multivariate data point an individual model (Model $1, 2, \ldots, n$), with respect to each dimension of the incoming multivariate data points. Thus, the forecast $\tilde{x_t}$ of an observed data point x_t is given by:

$$\tilde{x_t} = \begin{pmatrix} \tilde{x}_t^1 \\ \tilde{x}_t^2 \\ \vdots \\ \tilde{x}_t^n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^1 + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^1 \\ \sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^2 + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^2 \\ \vdots \\ \sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^n + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^n \end{pmatrix} \tag{A.4}$$

where $\nabla x_i = x_i - x_{i-1}$ and x_t^i denotes the datum corresponding to metric i ($i \in [1, \ldots, n]$) at time step t.

For each discrete model ensuing schema holds: Upon receiving the ground truth data value x_t, we incur the prediction loss for our forecast $\tilde{x_t}$. Thus the loss is $l_t^j(x_t^j, \tilde{x_t}^j)$ (loss corresponding to model j in time step t):

$$l_t^j(x_t^j, \tilde{x_t}^j) = (x_t^j - \tilde{x_t}^j)^2 = (x_t^j - (\sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^j + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^j))^2 \tag{A.5}$$

Given the loss, we update our model according to the previously discussed methods. Thus, the loss gradient is for updating the model's coefficients:

$$\nabla l_t^j[\gamma^t](x_t^j, \tilde{x_t}^j) = \begin{pmatrix} -2 \cdot (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-1}^j \\ -2 \cdot (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-2}^j \\ \vdots \\ -2 \cdot (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-w}^j \end{pmatrix}$$

As the model progresses through time, the historical data columns drop out of the window's tail, while the future data will be added to the head of the window. For an observed data point x_t, its differenced values are computed prior to being integrated into the window. The computation can be performed in an iterative manner, following the formula $\nabla^i x_t = \nabla^{i-1} x_t \breve{} \nabla^{i-1} x_{t-1},\ i > 0$. Thus, the model provides online computation, which is required for our use case. As this approach focuses on univariate processing of values, inter-dependencies between the dimensions are not captured. Next, we describe shortly how to capture those inter-dependencies and consequently establish multivariate computation through a single Online Arima model.

Multivariate Online Arima

As the ARIMA computation formulas are indifferent to the dimensionality of the input data (see Equation A.2), the model can provide also a multivariate forecast $\tilde{x_t}$. The Autoregressive part of the model is a linear combination of vectors, weighed by our model coefficients γ. The differenced vectors are given by $\nabla x_t = x_t - x_{t-1}$, where standard vector subtraction is performed. The consequent differences follow: $\nabla^i x_t = \nabla^{i-1} x_t - \nabla^{i-1} x_{t-1}$.

Again, upon receiving the observed data point x_t, we compute the prediction loss for our forecast $\tilde{x_t}$. The incurred loss is defined also by the squared Euclidean distance between the two vectors but captures now all the multivariate n dimensions. Let $l_t(x_t, \tilde{x_t})$ denote our incurred loss in time step t:

$$l_t(x_t, \tilde{x_t}) = \sum_{i=j}^{n} (x_t^j - (\sum_{i=1}^{w} \gamma_i^t \nabla^d x_{t-i}^j + \sum_{i=0}^{d-1} \nabla^i x_{t-1}^j))^2 \tag{A.6}$$

Thus, through the sum over the given dimensions the interdependencies are captured, and the loss gradient is consequently for the ONS procedure:

$$\nabla l_t[\gamma^t](X_t, \tilde{X_t}) = \begin{pmatrix} -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-1}^j \\ -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-2}^j \\ \vdots \\ -2 \cdot \sum_{j=1}^{n} (x_t^j - \tilde{x_t}^j) \cdot \nabla^d x_{t-w}^j \end{pmatrix}$$

The adaptions made to the original paper are the same as described above for the univariate Online Arima model.

Appendix B

Intervals for Identity functions and Threshold models

The following default intervals are integrated into the IFTM framework on which the hyperparameter optimization is applied (see Table B.1 and B.2). When not stated differently, the standard configurations of the Deeplearning4j[1] (version: 1.0.0-beta4) are used. Furthermore, for weight initialization the function Xavier [294], mean squared error as loss function and the output layer utilizes as activation function the linear identity.

[1] `https://deeplearning4j.org/`

Method	**Hyperparameter**	**Parameter intervals**
Gaussian aggregation	σ multiplier c	[0,10]
Exponential moving model	α	[0,1]
	σ multiplier c	[0,10]
Double exponential moving model	α	[0,1]
	β	[0,1]
	σ multiplier c	[0,10]
Sliding window aggregation	window size	{2,3,...,500}
	σ multiplier c	[0,10]

Table B.1: Intervals for hyperparameter optimization for threshold models.

Method	**Hyperparameter**	**Parameter intervals**
CABIRCH	initial threshold	0
	number of nodes	{2,3,...,100}
	logistic function decay - β	[0,1]
	logistic function decay - max decay	[0,1]
Univariate Arima	d	{1,2,...,10}
	mk	{2,3,...,500}
Multivariate Arima	d	{1,2,...,10}
	mk	{2,3,...,500}
Autoencoder	Encoding layers	{1,2,...,5}
	Decoding layers	{1,2,...,5}
	Learning rate	[0,1]
Variational Autoencoder	Encoding layers	[1,5]
	Decoding layers	[1,5]
	Learning rate	[0,1]
LSTM network	Layers	{1,2,...,10}
	Learning rate	[0,1]
Autoencoder with LSTM cells	Encoding layers	{1,2,...,5}
	Decoding layers	{1,2,...,5}
	Learning rate	[0,1]

Table B.2: Intervals for hyperparameter optimization for Identity functions.

Appendix C

Detailed Evaluation Results

Identity function	**Threshold model**	**TP-rate [%]**	**TN-rate [%]**	**FP-rate [%]**	**FN-rate [%]**	**AUC [%]**
CABIRCH	DEMM	58.12	90.16	9.84	41.88	**74.14**
CABIRCH	EMM	51.36	94.57	5.43	48.64	**72.97**
CABIRCH	SWA	56.14	87.95	12.05	43.86	**72.05**
CABIRCH	CA	56.04	85.82	14.18	43.96	**70.93**
AE	SWA	53.61	75.51	24.49	46.39	**64.56**
AELSTM	SWA	50.48	77.78	22.22	49.52	**64.13**
LSTM	SWA	51.33	76.62	23.38	48.67	**63.98**
MArima	CA	63.56	64.29	35.71	36.44	**63.93**
VAE	EMM	44.25	83.20	16.80	55.75	**63.73**
AE	CA	44.16	82.97	17.03	55.84	**63.57**
UArima	SWA	63.60	57.85	42.15	36.40	**63.41**
LSTM	DEMM	49.81	76.75	23.25	50.19	**63.28**
AE	EMM	39.75	86.23	13.77	60.25	**62.99**
AELSTM	DEMM	48.34	76.14	23.86	51.66	**62.24**
VAE	DEMM	46.42	78.04	21.96	53.58	**62.23**
AELSTM	CA	34.20	87.16	12.84	65.80	**60.68**
VAE	CA	37.15	82.54	17.46	62.85	**59.85**
LSTM	CA	34.34	84.67	15.33	65.66	**59.51**
UArima	CA	26.66	91.09	8.91	73.34	**58.88**
AELSTM	EMM	21.91	94.65	5.35	78.09	**58.28**
MArima	SWA	68.20	52.62	47.38	31.80	**55.83**
UArima	DEMM	21.85	88.10	11.90	78.15	**54.98**
MArima	DEMM	14.77	91.52	8.48	85.23	**53.15**
UArima	EMM	9.02	95.44	4.56	90.98	**52.23**
LSTM	EMM	7.20	96.14	3.86	92.80	**51.67**
MArima	EMM	8.17	94.36	5.64	91.83	**51.27**
AE	DEMM	1.72	98.27	1.73	98.28	**50.00**
VAE	SWA	5.88	93.41	6.59	94.12	**49.65**

Table C.1: Point-wise evaluation results showing the different rates (TP,TN,FP,FN) as well as the AUC value.

Identity function	**Threshold model**	**Precision**	**Recall**	F_1	**Accuracy [%]**
CABIRCH	EMM	71.03	51.36	59.61	**85.67**
CABIRCH	DEMM	60.52	58.12	59.30	**83.56**
CABIRCH	SWA	54.73	56.14	55.43	**81.40**
CABIRCH	CA	50.62	56.04	53.19	**79.69**
AELSTM	EMM	51.49	21.91	30.74	**79.66**
AE	DEMM	20.44	1.72	3.17	**78.38**
LSTM	EMM	32.60	7.20	11.79	**77.82**
UArima	CA	43.69	26.66	33.11	**77.82**
UArima	EMM	33.91	9.02	14.24	**77.64**
AE	EMM	42.82	39.75	41.23	**76.66**
MArima	EMM	27.30	8.17	12.57	**76.61**
AELSTM	CA	40.86	34.20	37.23	**76.25**
MArima	DEMM	31.13	14.77	20.04	**75.72**
VAE	SWA	18.79	5.88	8.96	**75.38**
VAE	EMM	40.58	44.25	42.34	**75.17**
AE	CA	40.21	44.16	42.09	**74.98**
UArima	DEMM	32.27	21.85	26.06	**74.46**
LSTM	CA	36.75	34.34	35.50	**74.30**
VAE	CA	35.56	37.15	36.34	**73.20**
AELSTM	SWA	37.08	50.48	42.75	**72.15**
VAE	DEMM	35.42	46.42	40.18	**71.53**
LSTM	SWA	36.28	51.33	42.52	**71.41**
LSTM	DEMM	35.72	49.81	41.60	**71.20**
AE	SWA	36.22	53.61	43.23	**71.00**
AELSTM	DEMM	34.44	48.34	40.22	**70.41**
MArima	CA	31.58	63.56	42.19	**64.14**
UArima	SWA	28.13	63.60	39.01	**59.03**
MArima	SWA	27.18	68.20	38.87	**55.83**

Table C.2: Point-wise evaluation results presenting precision, recall, F_1 score, and the accuracy.

Identity function	**Threshold function**	**C(time.avg, time.std)**
AELSTM	DEMM	0.9999
AE	CA	0.9991
CABIRCH	DEMM	0.9989
MArima	CA	0.9982
CABIRCH	SWA	0.9952
AELSTM	CA	0.9946
CABIRCH	EMM	0.9936
UArima	SWA	0.9935
MArima	SWA	0.9934
LSTM	CA	0.9914
VAE	CA	0.9850
LSTM	EMM	0.9820
LSTM	DEMM	0.9812
UArima	DEMM	0.9797
LSTM	SWA	0.9792
VAE	DEMM	0.9766
AELSTM	SWA	0.9766
AE	SWA	0.9727
AE	EMM	0.9710
CABIRCH	CA	0.9643
UArima	CA	0.9493
AELSTM	EMM	0.9470
VAE	EMM	0.9347
MArima	DEMM	0.8943
UArima	EMM	0.8627
MArima	EMM	0.8243
VAE	SWA	0.6805
AE	DEMM	0.6704
All	normal	0.9380

Table C.3: Correlations of the average time and standard deviation for all 28 IFTM models.

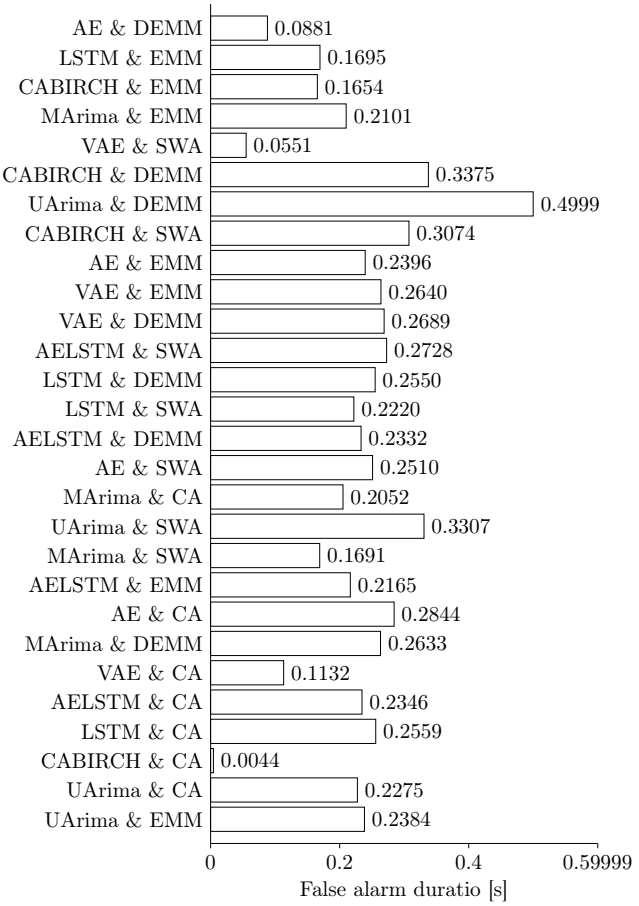


Figure C.1: False alarm duration for the 28 IFTM combinations.

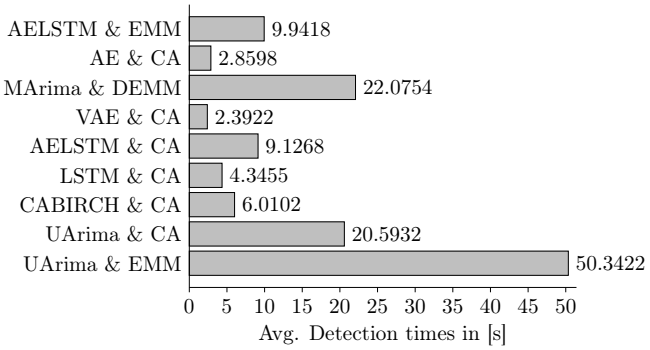


Figure C.2: Avg. Detection times in [s]. For lower than 100% detection rate.