A scalable machine learning system for anomaly detection in manufacturing

Zur Erlangung des akademischen Grades eines

Dr.-Ing.

von der Fakultät Maschinenbau

der Technischen Universität Dortmund

genehmigte Dissertation

M.Sc. Thomas Schlegl

aus

Starnberg

Tag der mündlichen Prüfung: 15.12.2023

1. Gutachter Prof. Dr.-Ing. Jochen Deuse

2. Gutachter Prof. Dr.-Ing. Rainer Müller

Dortmund, 2023

Abstract

Media reports on product recalls in the automobile industry have become a common occurrence for consumers. In fact, their frequency and the number of affected vehicles has been on the rise in recent years. The associated costs are an enormous economic burden to automobile manufacturers. Most recalls can be traced back to production faults. The ability of manufacturers to detect these faults early on during production and prevent a recall can be a competitive advantage. Aside from improved quality management processes, intelligent and automated analyses of the process data promises great potential that has remained largely untapped. This is because the challenges are immense: the amount of data is enormous and the data patterns characteristic of a process fault are unknown. A promising approach is the use of Machine Learning (ML) to scour this data for the metaphorical needle in the hay stack. The goal is to use algorithms that learn to distinguish between normal and anomalous process behavior and warn the process expert if an anomaly is detected. Researchers in both industry and academia have been trying for years to establish such ML systems in manufacturing. However, most ML projects fail before reaching the productive phase. The few systems that do make it into production require a huge amount of resources for system operation and add little net economic value to the business. This has lead to the widely accepted realization that, while it is relatively easy to develop ML prototype systems that exhibit sometimes bafling results, it is incomparably more difficult to scale these into productive systems that monitor hundreds or even thousands of processes. Surveys of ML experts confirm, that often unnecessarily complex algorithms are used for these systems and that the long-term complexity that these design choices entail for system operability are either ill understood or not considered.

The goal of this thesis is the development of an approach to train scalable ML models for anomaly detection in manufacturing process data. The training process for model initialization and adaptations must be highly automatable to allow for a structured and orchestrated scaling process. This should allow to reduce the complexity on a system level and facilitate long-term operability. Due to the practical problem statement and applied ML research, the Design Science research methodology was central to this thesis. This research paradigm focuses on generating new knowledge by iterative development and field testing in an applied setting. Based on literature, two promising techniques were identified that served as the starting point for the development. On the one hand are Deep Learning models (DL) whose prediction accu-

racy outperforms alternate approaches in most benchmark studies, irrespective of the field of application. On the other hand, are Data Mining techniques (DM) where considerable advances have been made in recent years that improved the performance and technical implementation of powerful DM algorithms. The majority of the thesis focuses on a bottom-up evaluation and iterative field-testing of both techniques in parallel development cycles to arrive at a scalable approach for constructing large-scale ML systems in production. Their scalablility was subsequently evaluated under real-world condition during a 20-week evaluation phase. To ensure the comparability of both approaches, a purpose-built ML management software was used to ensure a structured process for training, scaling and adapting models. The number of models required for monitoring the production system of the DM/ML approach was nearly half compared to the DL approach. The DM/DL approach allowed to trade a short-term increase in resources required for model training for improved long-term system maintainability and can be considered scalable in both relative and absolute terms. This conclusion is supported by developer reports of large-scale ML systems in production that are comprised of low-complexity DM/ML models.

Acknowledgement

First and foremost I want to thank my brother Stefan. He served as an invaluable sparring partner and lent his unconditional support throughout the entirety of my doctoral studies. The research summarized in this thesis would not have been possible without him.

I would like to thank my doctoral supervisor Prof. Dr. Jochen Deuse for ensuring the scientific rigor of my work while at the same time giving me the utmost freedom to pursue my research. I am also grateful to my industry supervisor Sebastian Mayer, who I could always rely on for support and have come to view as a personal mentor. Both have shaped me as much as the research work itself and have made me grow as a person.

I would like to thank my friend Dr. Daniel Carton for focusing my efforts on the publication of my research work. He also told me, that ”a good dissertation requires the willingness to sacrifice personal relationships.” I am now engaged to my then girlfriend Samirah - which I try to convince myself is a testament to her courteous and accepting nature rather than the quality of this dissertation. I will let the reader be the judge of the latter.

Lastly, I would like to thank my family for their unwavering encouragement and emotional support, particularly my parents Ulrich and Sabine, my sister Christina as well as my grandparents Ottmar and Elisabeth.

Contents

Nomenclature

Abbreviations

AI	Artifical Intelligence
ANN	Artificial Neural Network
APCA	Adaptive Piece-wise Constant Approximation
AUC	Area Under the Curve
CASH	Combined Algorithm and Hyperparameter Selection
CNN	Convolutional Neural Network
DL	Deep Learning
DM	Data Mining
DP	Distance Profile
DSR	Design Science Research
DTW	Dynamic Time Warping
FMEA	Failure Mode and Effects Analysis
GRU	Gated Recurrent Unit
HPO	Hyperparameter Optimization
ICLR	International Conference on Learning Representations
ICML	International Conference on Machine Learning
IR	Information Retrieval
LSTM	Long Short-Term Memory
LVM	Latent Variable Model
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MLM	Model Lifecycle Management
MLOps	Machine Learning Operations
MLP	Multi-Layer Perceptron
MNIST	Modified National Institute of Standards and Technology
MP	Matrix Profile
NeurIPS	Neural Information Processing Systems
NHTSA	National Highway Traffic Safety Administration

NN	Nearest-Neighbor
OEM	Original Equipment Manufacturer
PAA	Piece-wise Aggregate Approximation
PRF	Pseudo Relevance Feedback
RNN	Recurrent Neural Network
SAX	Symbolic Aggregate Approximation
SME	Subject Matter Expert
SVM	Support Vector Machine
VAE	Variational Autoencoder

Symbols

L	Loss term
Θ	Threshold
A	Subsequence set
a	Activation vector
b	Bias vector
C_S	Snippet candidate
D	Distance profile
h	Hidden state vector
J	Similarity join
P	Matrix profile
Q	Query subsequence
S	Snippet
T	Time series
w	Weight vector
x	Input vector
y	Output vector
z	Probablistic latent vector

List of Figures

List of Tables

1 Introduction

Product recalls in the automotive industry are notorious for their enormous scale and cost. The costs of replacement or repair, legal litigation and loss in sales can be devastating. In 2016, over 50 million recalled vehicles were reported to the U.S. National Highway Traffic Safety Administration (NHTSA) in the Unites States alone [11], costing automobile manufacturers over 22 billion dollars in warranty accruals [12] - approximately equivalent to the annual R&D expenditures of the industry [13]. While this was the worst year on record, both in terms of the number of recalls and total number of affected vehicles, it is indicative of a broader trend [14, 15]. Over the past decade, more than 30 million vehicles have been recalled in the U.S. per year on average [11]. In any given year more vehicles are recalled than sold [16]. There are structural reasons why these high recall rates persist. Increasing product complexity and shorter development cycles, driven by customer demand and regulatory requirements, as well as growing competition from new emerging manufacturers force original equipment manufacturers (OEMs) to cut costs in order to stay competitive. Their widespread response has been to adopt common part strategies across their product portfolio and outsource business activities to an increasingly globalized value chain. This greatly increases the number of potential sources and amplifies the repercussions of quality defects. The vast majority of these sources can be classified as design flaws and manufacturing faults [17], whereby the latter account for nearly 60 percent of all recalls [18]. There is, therefore, an enormous economic incentive for OEMs to develop and implement intelligent measures to detect and avoid manufacturing faults in their operations.

1.1 Background and Motivation

Background The starting point for a recall is usually a quality issue that is detected along the value stream of the OEM or ”in the field” by customers and service shops after the vehicle has been delivered. To investigate reported quality issues, OEMs often conduct root cause analyses based on the process data recorded during production. The aim is to identify characteristic patterns in the data that can be associated with the quality issue in the field. If the production data contains a robust indicator of the fault, it can be used by OEMs to recall only affected vehicles in a precision recall. While this is obviously desirable to minimize the economic fallout, it also means that the recall could have been avoided if manufacturers had been able to detect the pattern preemptively. The problem is that such patterns are initially unknown - after all, if

they had been known, measures would have been put in place to detect them. Thus, preventing recalls requires manufacturers to detect patterns that they know exist without knowing what they look like - essentially to detect ”known unknowns”. Standard control systems rely on statistical thresholds to ensure process conformity and detect *known* patterns through a set of if-then-rules [19]. These systems are unsuited for reliably detecting *unknown* patterns. Instead, a monitoring system is required that raises a warning if an anomalous pattern is detected that deviates from normal process behavior. It is a challenge in its own right to design a system that detects these anomalies - after all, the distinction what is normal and what is not is often difficult to define in precise terms. To complicate matters further, an anomalous pattern only ever *may* indicate a fault. Just like not every fault presents itself as an anomalous pattern in the data, not every anomalous pattern is indicative of a fault. This distinction between anomalies that are indicative of a fault and those that are not cannot be inferred from the data alone, but requires the input of subject matter experts (SMEs). While SMEs may not be able to define a set of general classification rules, they are often able to judge if a concrete object is critical or not (”I know it when I see it”). The system must therefore *learn* to make this distinction based on user feedback and do so efficiently to avoid overburdening the user with a flood of irrelevant warnings [20].

Artificial intelligence (AI) is the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as pattern recognition and decision-making [21]. In principle, it should be possible to apply this technology in the manufacturing domain and develop an adaptive anomaly detection system. In particular, the AI-subfield of machine learning (ML) provides the necessary tools to enable the system to adapt its ability to detect meaningful anomalies based on user-provided feedback. ML is the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data [1]. The automotive manufacturing industry is often credited with being particularly well-suited for the application of ML technology [22, 23]. This is due to the high degree of automation and machine connectivity that results in the availability of large amounts of data as well as a high-volume, high-value product that allows to quickly amortize development costs. For these reasons, researchers have been working on transferring the popularized successes of ML applications to the manufacturing domain for many years. Compared to other ML applications

[1] Oxford dictionary

that seek to improve quality control (which accounts for the majority of AI-related expenditures in the manufacturing domain [24]), an adaptive anomaly detection system is arguably one of the most promising applications of ML technology. Rather than seeking to cut costs by replacing existing quality assurance systems with an ML system that is trained to do the same task, the system *extends* the capability of manufacturers. Consider the often-cited case of predictive quality, where the aim is to detect patterns in the manufacturing data that are known to correlate with manufacturing defects (or lack thereof). If these correlations are reliable, downstream testing procedures could be shortened or avoided. However, this is only possible if the system is both accurate and reliable. Even if this is the case, cost savings are limited to the investment that would otherwise be necessary to provide sufficient test capacity (minus the cost of developing and operating the ML system). In contrast, an anomaly detection system seeks to prevent field recalls that OEMs currently have no effective way of preventing, seeking to *extend* rather than *replace* existing quality assurance measures.

Motivation The idea of developing ML systems for applications in the manufacturing domain is not new. However, despite decades of applied research and countless prototypes that have demonstrated the potential of industrial ML systems for a range of applications (including anomaly detection) [25, 26, 27, 28, 29, 30], the large-scale adoption of these systems has so far proved evasive. This phenomenon is not limited to the manufacturing domain [31]. While more than 90 percent of companies invest in AI [32], only around one fifth have deployed ML systems into production [33, 23] with only ten percent of these systems adding significant economic value to the business [34]. For the majority of cases the cost for system development and operation presumably exceeds its business value. This is corroborated by a poll conducted among ML practitioners across various industries, that shows that the overwhelming majority of ML models intended for productive use are never actually deployed. The results of the poll are shown in figure 1.

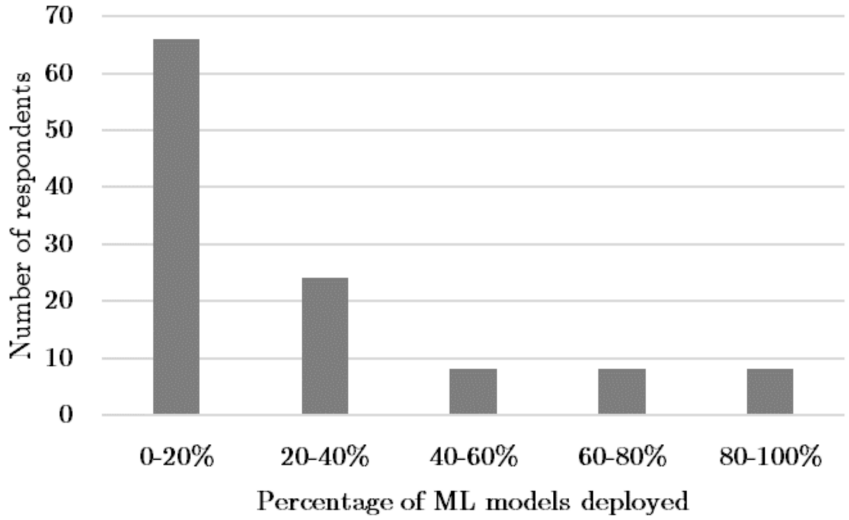


Figure 1: Result of a poll among 114 ML practitioners about the percentage of ML models deployed to production [1]

How can the touted successes of small-scale pilot systems be reconciled with the apparent inability of established companies to build on this success and move these systems into production? It is often assumed, that the adoption of new technologies follows a linear path from fundamental research to applied research to commercialization [35]. However, this process is considerably more complex in practice. In the case of real-world ML systems, the reason for this slow technology transfer is twofold. First, is the often overlooked fact, that the operation of a productive ML system is *considerably* more complex and difficult than its development. While developing and deploying ML systems is relatively fast and cheap, maintaining them over time in production is difficult and expensive [36]. This has been known in software engineering for a long time. The failure to consider the challenges associated with system operation during the development phase quickly results in a system that is inoperable when taking into account economic factors. This is compounded by the pressure to deliver results in a short amount of time, which is particularly pronounced for industry projects [37]. This results in technical debt - a common phenomenon in software engineering that refers to the compounding effect of suboptimal design considerations during the development stage on the long-term cost of rework and maintenance. The same reasons that make this true for software systems apply to ML systems. In fact, ML systems are usually *considerably* more complex than conventional software systems, requiring additional processes for model life cycle management, orchestration of training pipelines, tracking data dependencies, etc. The added complexity of real-world ML-systems is often described as a special capacity of these systems to incur *hidden* technical debt [36]. This

is further compounded by the fact that the development of ML systems is often done by people with a background in engineering or mathematics that have limited experience in professional software development [38]. Consequently, these systems often consist of a patchwork of legacy code and glue-code snippets. Such systems are extremely difficult and costly to maintain. To set up and operate these individual subsystems requires a broad range of expertise [39]. This has led to the rise of a new discipline that combines expertise in the area of machine learning, data engineering and software engineering, called MLOps. As can be gleaned from figure 2, MLops has been gaining momentum in recent years, which shows, that people are increasingly aware of the challenges of ML system operation.

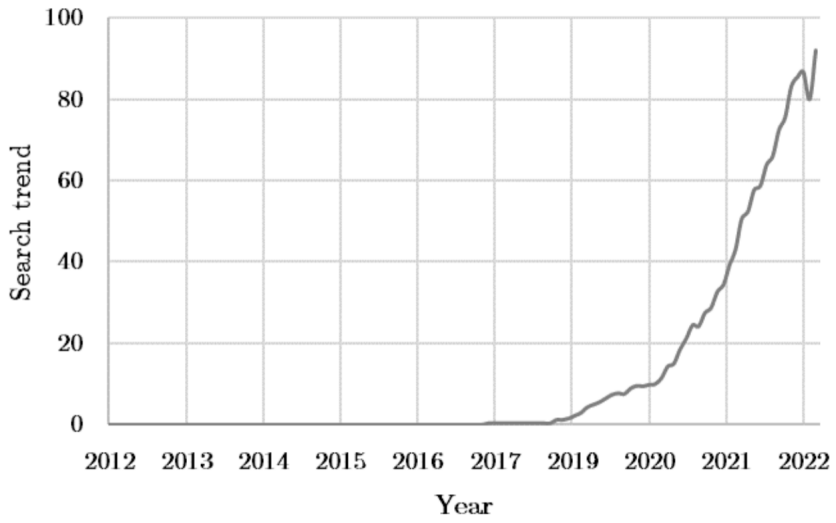


Figure 2: Google search trend of the term ”MLOps” as a proxy for the growing focus of researchers and practitioners on the challenges of ML system operation.

The second major reason for the slow technology transfer is economics. While most companies struggle to deploy ML technology into production, tech companies have been offering large-scale ML-based products and services for years [40, 41, 42, 43]. Clearly, neither the maturity nor accessibility of the technology is the problem. OEMs themselves have commercialized ML-based product features like voice recognition and advanced driver assistance systems at scale. Most notably, they are investing huge sums into the development of autonomous driving technology. These systems are undeniably more complex than an anomaly detection system, yet most researchers expect level four autonomy to be made available to customers by 2030 [44]. To understand this asymmetry, one must consider the differing business value of these ML systems. The value of an industrial ML system is determined by its ability to improve the bottom line of manufacturing operations through cost savings. These costs are dominated by variable

costs that are either highly optimized or outside the control of the OEM. Between 40 and 50 percent of a passenger vehicle's retail price is attributable to material costs [45], roughly half of which are in turn attributable to the price of steel. A ten percent drop in the price of steel, therefore, increases profit margins by up to two percent [46]. To realize the same increase in profit margins through internal cost saving measures would require cutting the R&D budget or capital expenditures by one third (both R&D and depreciation account for roughly 6 percent of costs [47, 48]. In contrast, autonomous driving technology could conceivably contribute more to the top line of an OEM than all possible cost saving measures combined could contribute to the bottome line. This rough breakdown of the cost structure of automobile OEMs shows, that the economic potential of industrial ML systems is inherently limited. It follows, that the resources that managers are prepared to commit to the development and operation of these systems is equally constrained.

In summary, while researchers have demonstrated the potential of ML systems in the manufacturing domain in general and for anomaly detection in particular, OEMs struggle to move these pilot systems into production. This is because ML system operation is considerably more difficult than the development of pilot systems may suggest. The additional complexity of ML systems compared to conventional software systems results in a propensity to incur hidden technical debt that degrades system operability. Despite the convincing case for adopting AI technology for large-scale anomaly detection in manufacturing, the resources available for maintaining and operating the system are inherently constrained. On the one hand, this requires establishing effective MLOps practices within the business. This is largely an organizational responsibility of hiring the right people to do the right things. On the other hand, system operability must be the central design tenant during the development of such a system. This thesis was motivated by the latter question of *how* to approach the development of a machine learning system for large-scale anomaly detection in manufacturing.

1.2 Research topic

Research question Production systems are highly controlled and optimized systems, which means that faults are exceedingly rare, isolated events. This means that detecting an (interesting) anomaly is comparable to the figurative search for a "needle in the hay stack". The system must be capable of monitoring a large number of processes in parallel to increase the likelihood of detecting meaningful anomalies. Additionally, the ML models must be continu-

ously retrained based on user feedback to remain effective. Ensuring the ability to continuously adapt the system at scale in an environment of severely constrained resources is extremely challenging. To ensure system operability, the manual effort required for model training must be sufficiently low, requiring a high degree of orchestration of the training process. This, in turn, requires suitable ML algorithms.

The models that make up the ML system have a fundamental effect on the complexity and, therefore, operability of the system (complex systems are inherently more difficult to maintain [49]). To minimize system complexity data scientists should select suitable ML algorithms to train these models. However, there are no established best practices for algorithm selection [50]. Countless comparative studies of the same algorithms continue to be published in literature [51, 52, 53]. In practice, the selection process is often unsystematic and highly dependent on the preference and experience of the developer [54]. This unstructured process of algorithm selection and model training often results in the adoption of unnecessarily complex models that can overwhelm development teams [50]. Managing complex models consumes huge amounts of developer resources and quickly becomes intractable at scale. Excessive model complexity is perhaps the most underappreciated impediment to the operability of real-world ML systems and will often impede the transition from a pilot to a (feasible) productive system.

Definition 1 *The research goal of this thesis is the development of a framework that allows data scientists in the manufacturing domain to build an ML system for anomaly detection that remains operable at scale. The central research focus is the development of ML algorithms for training and adapting ML models that are easy to manage and maintain. This avoids an unstructured algorithm selection process and allows data scientists to devote their time to scaling and optimizing the system.*

In the context of this thesis, the following definition of a framework is taken from the field of software engineering:

Definition 2 *A framework is a semi-complete application. A framework provides a reusable, common structure to share among applications. Developers incorporate the framework into their own application and extend it to meet their specific needs [55].*

From a practical perspective, a framework is a system providing generic functionality that can be selectively changed to develop a specific application. It is of a technical nature and must

therefore be distinguished from a high-level conceptual framework. The framework is intended to serve ML practitioners in the manufacturing domain as a support structure that facilitates the development of large-scale anomaly detection systems irrespective of the particular manufacturing setting (albeit with some constraints that are discussed in the research scope in the next section 1.2). It provides data scientists with ML algorithms that can be selectively applied to train both unsupervised and semi-supervised models as well as a process for managing the model life cycle that includes training, adapting and versioning of these models.

Research field and scope This work falls in the field of data science research. Despite its widespread use there is no consensus on the definition of the term data science. However, it is undoubtedly a field of study in its own right: universities offer degrees in data science, there are job listings for data scientists and researchers get funded to do data science research. Broadly speaking, it is an interdisciplinary field that is concerned with extracting actionable insight from large data sets using algorithms to solve real-world problems [56]. It incorporates elements from (software) engineering, computer science, mathematics (in particular statistics) and data management, among others [39, 38]. This interdisciplinary nature is shown in figure 3. As discussed, machine learning is concerned with (a) the development of computer systems that use (b) algorithms and statistical models to draw inferences from data. This corresponds to the primary objective of computer science and statistics, respectively. Computer science is itself an area of academic research, while software engineering focuses on programming and software design to solve practical problems. Data science research combines these sub-fields with a clear focus of providing a practical solution to real-world problems.

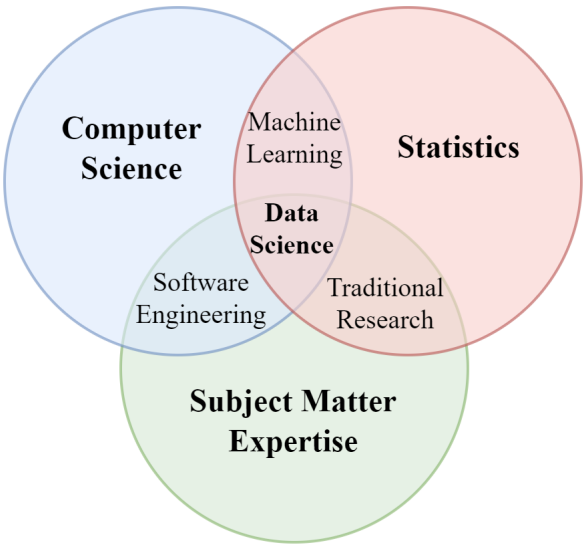


Figure 3: The field of data science is an interdisciplinary field of traditional research, machine learning and software engineering [2].

The focus on applied research in a real-world setting is a central aspect of this thesis and reflected in the research methodology discussed in the following section 1.3. The goal of the research is to generate knowledge that is immediately relevant to data scientists in the manufacturing domain. The work combines existing knowledge from literature with insights gained from rigorous scientific experiments and practical experience gathered during the development and operation of a real-world ML system. The ML system was developed, deployed and operated in the automotive assembly industry with the aim of detecting (interesting) anomalies in tightening processes of threaded fasteners. Tightening defects are some of the most common reasons for manufacturing-related recalls in the automotive industry [18] and thousands of threaded joints are used in the assembly of an automobile and its components - thus, this setting is an ideal case study of the potential and challenges of a large-scale industrial ML system for anomaly detection. Below is a summary of the most important aspects that are considered within or outside the scope of this thesis.

- Time series data is widespread in the manufacturing domain, especially for process data. This requires ML models that are capable of pattern recognition in time series data.
- The interdisciplinary nature of the data science field means that a broad range of knowledge is required. The hunt for talent that combines all of this knowledge has been likened to that of a unicorn [39] - something that one may wish for but does not exist. Therefore, these responsibilities are often distributed across multiple functions in an organization.

The establishment of structures within an organization that facilitate close collaboration between these functions for an effective MLOps process is not considered. This is predominantly a managerial challenge and lies outside the scope of this thesis.

- Monitoring systems that communicate directly with machine controls often require real-time data processing and are heavily constrained regarding the availability of network bandwidth. This low-latency, low-bandwidth requirements have fueled the shift from centralized computing towards distributed edge computing or embedding the model directly within the system. These considerations regarding the topology of the system are not considered.

1.3 Research methodology

1.3.1 The importance of applied ML research

The science of machine learning is, above all, a science of engineering, dedicated to creating knowledge about the design and construction of computer programs that use data to build practically useful models [57]. However, researchers often focus on incremental improvements of algorithms on benchmark data sets. This is evident from the large number of published research papers that present algorithms that outperform existing state of the art methods by narrow margins. These purported improvements are often small and frequently fail to translate to real-world settings [31]. The performance of developed algorithms often rapidly diminish once deployed into production [31]. This is not limited to research in the manufacturing domain. For example, while current state of the art models for machine reading comprehension (MRC) (a sub-field of natural language processing) have purportedly ”surpassed human performance” [58] on a narrow set of benchmark datasets, these results do not generalize well to new data. Even the most advanced MRC-models continue to make basic mistakes that show that they are a long way from true reading comprehension [59]. Why is there such a consistent discrepancy between reported performance in machine learning research and real-world applications? The innate bias of the benchmark data sets used to compare the novel methods play a significant role and have been increasingly criticized in recent years as out of touch with real-world data [60]. However, the main reason behind this trend is attributable to a shift in the research culture over the past decades [61]. The growing political, financial and public interest in the field of machine learning has led to a rapid increase in the size of the research community. This has fueled an increase in the number and frequency of published papers and caused growing competition among ML

researchers. Figure 4 shows the number of papers accepted for publication by the Conference on Neural Information Processing Systems (NeurIPS), the number of first-time authors and the average number of papers within an author's first five-year period over the past three decades. NeurIPS is widely credited with being one of the most prestigious venues for ML researchers, which is why it is selected here as a proxy to showcase a broader trend.

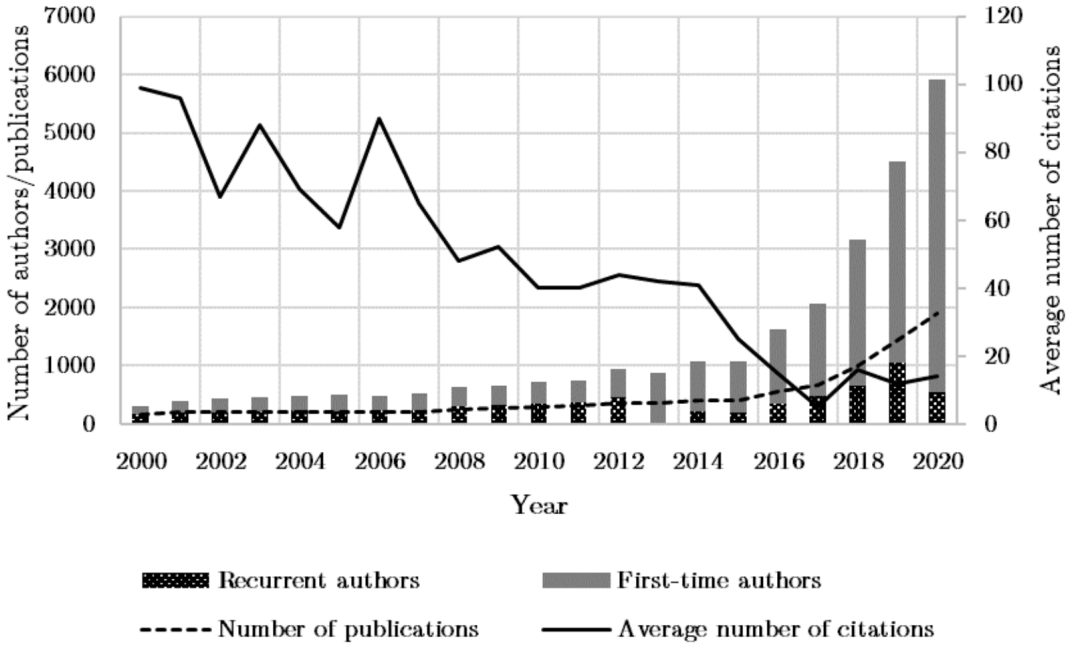


Figure 4: The number of accepted papers, their authors and average number of citations of paper published in the NeurIPS conference from 2000-2020.

This figure indicates, that a growing number of (first-time) researchers are publishing an exponentially growing number of papers. While this gives the superficial impression of increased productivity, the average number of citations (and relevance) of the papers has declined steadily. Faced with this flood of papers, many researchers attempt to increase their odds of getting published by focusing on quantitative (marginal) improvement of the state of the art. This results in a narrow focus on model performance on benchmark data sets at the expense of their real-world applicability. In some cases, researchers (knowingly or not) tweak models to beat the state of the art by exploiting spurious patterns in data [62]. A Meta-review of deep learning research demonstrated, that many published improvements, claimed to be due to innovations in network architectures, were actually due to simple hyperparameter tuning [63]. As a consequence, renowned experts in the field, including board members of NeurIPS and ICLR, have recently called for revision of the publication process in the field of ML. While this is predominantly an academic discussion, the conditioning of researchers to prioritize marginal improvements on

benchmark datasets over real-world problems incurs a very real opportunity cost. To make real contributions, researchers must solve real-world research questions and focus on obstacles that impede the practical applicability of machine learning. In a nutshell, applying ML to real-world problems requires applied ML research. Naturally, the transition between applied research and fundamental research is gradual. Instead of relying on generic definitions, it is much easier to compare their *differences* along a number of dimensions that characterize the type of research. The most important distinguishing features are summarized in table 1

	Fundamental research	Applied research
Purpose	The extension of knowledge is an end in itself	The goal is to improve understanding of a problem to contribute to its solution
Scope	Narrow research scope with tightly focused research question	high-level issue that raises multiple broader research questions
Context	Funded by grants and executed in an academic setting with lower cost and time pressure	Funded by contracts and executed in a field setting with inflexible goals
Method	Rigorous experimental design and control of variables to establish sound causal relationship	Focus on the practical effects of interest in a complex environment

Table 1: Comparison of fundamental and applied research according to Hedrick et al. [10].

Based on this summary, the research in this thesis clearly falls within the domain of applied research. The distinction between fundamental and applied research is one of degree and not of kind. However, it has important implications for the design and execution of the research. Specifically, the set of methods and best practices that can and should be used to address the research questions are generally much broader than in fundamental research. The research strategy that describes how research is to be undertaken and what methods are to be used is commonly referred to as a research methodology. A methodology is different from a method - the former is a contextual framework that is used to plan the research during the initial stages, whereas a method is a technique of data collection and evaluation that is laid out by this framework [64]. Essentially, the methodology is the justification for using a particular research method. This distinction may seem somewhat blurry and at times incidental - however it helps the reader get a sense of how the research in this thesis was carried out and the motivation behind its structure.

1.3.2 Design science research

The research presented in this dissertation is based on the design science research (DSR) methodology. DSR is centered around the development and application of artifacts as a means

of creating knowledge in a specific domain [65]. An artifact can be any logically coherent research outcome like an algorithm, system design, process model etc. [66]. The main goal of creating this knowledge is to enable the design of solutions for real-world problems. DSR is constructive research, compared to more conventional explanatory research that is focused on establishing correlations between observed variables [67]. It is intended to bridge the gap between research and practice [68] whose 'theory-with-practical-implications' research strategy often fails to produce results that are of practical interest [69]. Essentially, it is the theoretical manifestation of "learning by doing". DSR subdivides the research activity into three distinct research cycles depicted in figure 5. The relevance cycle ensures the *relevance* of the research by connecting the application domain with the design science activities. It initiates the research by providing the research questions and requirements and serves as a feedback-loop for field testing the designed artifacts in a real-world environment [3]. If the artifact does not satisfy the initial requirements, or these requirements turn out to require adjustment, the design cycle is reiterated. The rigor cycle connects this design cycle with a knowledge base that contains expertise and state-of-the-art artifacts from the application domain to ensure the scientific *rigor* of the research. It is this cycle that distinguishes design science from the everyday practice of building artifacts [69].

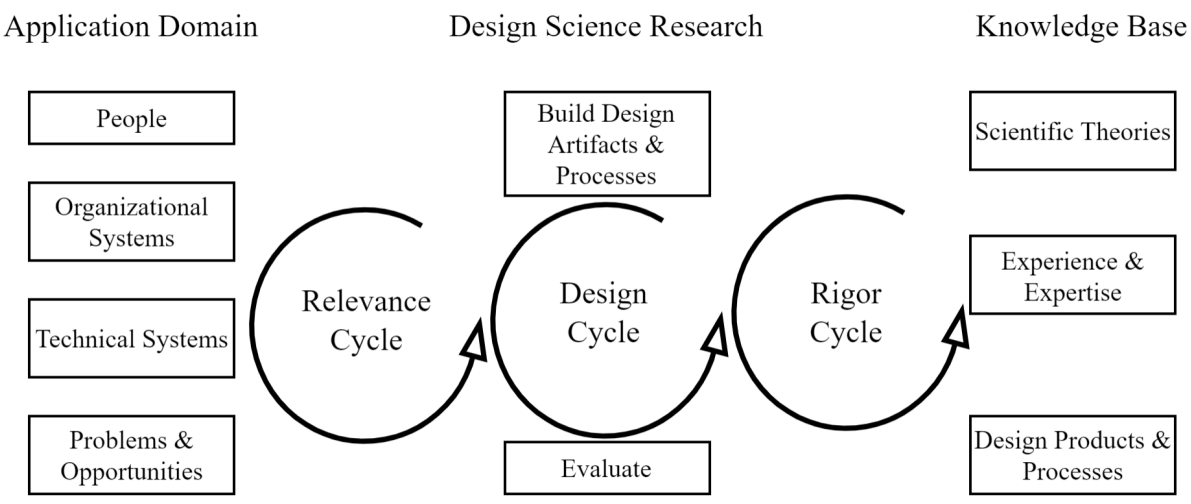


Figure 5: Design science cycles according to Hevner et al. [3]

Design science is most widely applied in the engineering and computer science disciplines and particularly widespread in the field of information system. However, in recent years, it has been adopted by researchers in various field such as the social sciences [64]. Renowned research insti-

tutions like the MIT's Media Lab, Stanford's Centre for Design Research and Carnegie-Mellon's Software Engineering Institute use this methodology. A majority of the research proposals submitted to the U.S. National Science Foundation in the field of computer and information science and engineering use a DSR methodology. Due to the thematic overlap of this dissertation with these disciplines as well as the interdisciplinary research context discussed in section 1.2, design science was the research methodology of choice for this work.

What does this imply for the research and structure of this dissertation? The cyclical approach encouraged by the DSR methodology reflects the iterative and non-linear work that characterized this research. The first relevance cycle initiates the research by defining the practical requirements laid out in the following section 2. Most state-of-the-art approaches either did not meet these requirements or were not reproducible (this is discussed in detail in section 3). Based on expert opinion and iterative testing, two promising approaches emerged. As a result, two parallel design cycles were initialized for the development of separate algorithms. The rigor cycle continuously integrated research results of other research groups as they were published. For example, the algorithms presented in section 4 is based on research that was published while the work was ongoing. The algorithms were quantitatively evaluated in experiments on real-world and synthetic data sets. This is a common method for evaluating algorithms in scientific literature [70]. Most research does not go beyond this step. However, most design impulses came from the relevance-cycle of field-testing in the manufacturing setting and feedback from SMEs. The results and final design considerations of each approach are consolidated in section 4 and section 5. Additionally, these relevance cycles initiated a third design cycle for the development of a broader framework to integrate the algorithms on a system level. The framework and results are summarized in section 6.

2 Challenges and requirements

This section discusses the interdependent challenges and requirements of realizing a large-scale anomaly detection system in the manufacturing domain and the shortcomings of attempts to realize such a system that are published in literature. The first three subsections deal exclusively with these challenges and requirements. The first subsection discusses generic high-level system requirements relating to the operation of large-scale productive ML systems. Note, that the term *productive* system is used as is common in the field of software development i.e. a system

that is used live by stakeholders. The second subsection focuses on the ML problem of anomaly detection. This is an essential discussion that sets the stage for much of the work presented in this thesis. The third subsection focuses on the particular challenges associated with pattern recognition in time series data. The final subsection analyzes the vast amount of literature on applying ML technology for anomaly detection and fault detection in the manufacturing domain and discusses the shortcomings of existing attempts to realize a large-scale system.

2.1 Operating large-scale machine learning systems

This sections discusses the most important non-functional requirements associated with the development and operation of large-scale ML systems in an industrial context. Non-functional requirements describe properties of the system relating to its operation (compared to functional requirements, that describe a specific behavior or function of the system) [71]. These are shared by virtually all real-world ML systems to some degree.

2.1.1 System scalability by extension

Although the general notion of what is meant by scalability seems clear, its lack of a precise definition has been criticized for decades. People will often refer to a system as being scalable or not - however, this is a gross oversimplification. Almost any system will scale to some degree and, just as importantly, no system scales indefinitely [72]. The informal definition of the term that most readers will be familiar with serves as a starting point [73]:

Definition 3 *Scalability is the ability of a system to continue to function with acceptable performance when the workload has been significantly increased.*

Of course, what constitutes *acceptable* performance depends entirely on the setting. This definition lacks the necessary rigor to be of much practical use. It makes no mention of *how* the system's capacity to handle the workload is increased and, more importantly, the *resources* required to make this increase possible [72]. If increasing system capacity requires an unfeasible amount of resources, we would not consider this system scalable in a practical sense, even if this is technically possible. In the context of this thesis, the *workload* of the system is the number of manufacturing processes that must be monitored. This can range from a handful of processes for small (pilot) systems to hundreds or even thousands of processes for large systems required to monitor entire production systems. As this number is increased, the capacity of the system to monitor these processes must scale accordingly. Extending the system's capacity can be

realized in one of two ways, namely by (1) adapting existing models or (2) deploying additional models to monitor these processes. This can be thought of as similar to scaling a computer cluster, whose total computing power is increase by either upgrading existing machines in the cluster (vertical scaling) or adding additional machines to the cluster (horizontal scaling). This same terminology can be adapted for scaling of ML models and is schematically depicted in figure 6.

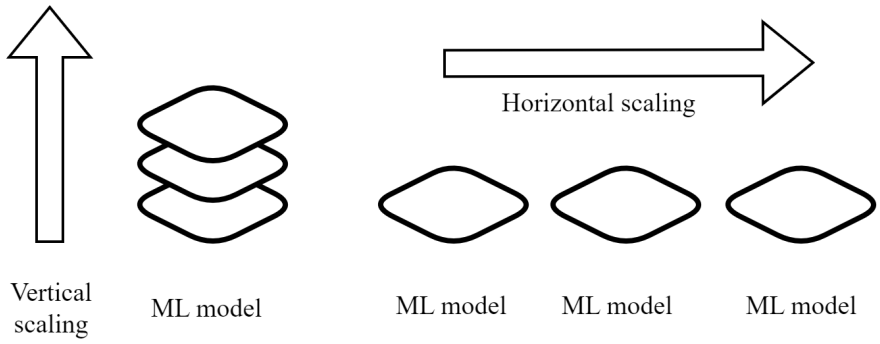


Figure 6: Horizontal vs. vertical scaling of ML models to increase system capacity.

The discussion so far focused on the definition of system workload as well as ways to increase the system's capacity to handle it. The obvious next question is which approach is preferable - should models be scaled horizontally or vertically as the workload increases? This questions leads to the second important consideration, namely the *resources* required for scaling the system. The short answer is that this largely depends on the nature of the processes - neither scaling approach is per se better than the other (in fact, a combination of both is often preferable in practice). Consider the extreme cases of two systems that monitor the same number of processes, where system A consists of a single vertically scaled model and system B consists of multiple horizontally scaled models that each monitor a single process. All knowledge learned by system A is consolidated in a single model, whereas the same knowledge is distributed among multiple models in system B. The ease of scaling these systems to a new process depends on how similar that process is to those already monitored by the system. If the new process is sufficiently similar, system A can be scaled without any changes to the model. If the process is different, however, the model must be adapted. The difficulty lies in retraining the model without negatively affecting its performance for other processes. Ensuring this incremental improvement is challenging. Optimizing the performance for one process may adversely affect that of other processes. This is known as the CAKE-principle ("changing anything changes everything") and may require lengthy tuning of model hyperparameters. As the system is

scaled, these data dependencies grow as well, magnifying the problem. These dependencies and the associated problems can be avoided by scaling models horizontally. The simplest way to do this it to train a completely new model ”from scratch”. This is by far the fastest way to getting a model into production. However, if the process is similar to existing processes, manual feedback from SMEs is required to ”relearn” knowledge that is already incorporated in other models. This wastes the constrained resources of SMEs and prolongs the time until the model is able to detect meaningful anomalies. This can be avoided by starting with the model of the most similar process, essentially creating a ”branch” of an existing model. This transfers previously gathered knowledge to the new model and is a form of transfer learning. The advantages and disadvantages of horizontal and vertical scaling are summarized in figure 7.

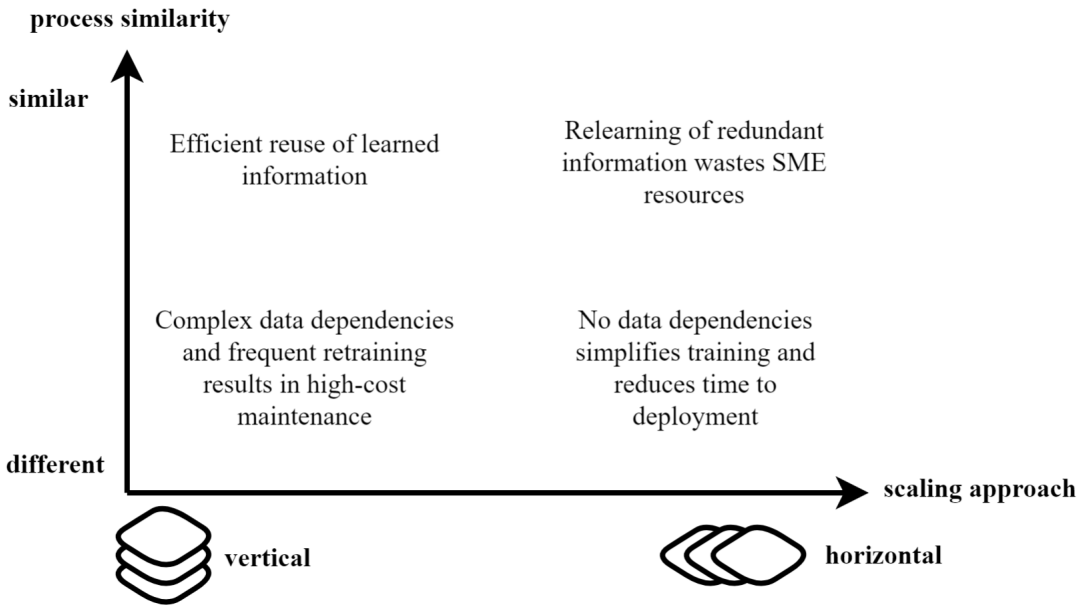


Figure 7: Advantages and disadvantages of horizontal vs. vertical scaling of ML models.

Essentially, the choice is determined by a trade-off between the speed of deployment, the required input of SMEs for data annotation and the complexity of the resulting models. This, in turn, depends on the similarity between the processes. In most settings, some processes will be very similar while others are different. For example, process data of insertion operations is vastly different from that of tightening operations, which can be subdivided into dozens of sub-categories depending on the machine, fastener sequence, assembled part etc. Thus, a combination of horizontal and vertical scaling may be preferable. However, determining which processes can be grouped together is extremely challenging. Asking SMEs to manually group processes that *appear* to be similar is not reliable. This is because the question of similarity is

one of degree not of kind i.e. judging whether two objects are objectively similar requires both a similarity measure and a threshold - the selection of which is challenging and often task-specific [74]. This is why clustering time series data is so difficult [75]. Therefore, the initial decision to scale vertically/horizontally is only an initial best guess and may need to be revised later on as more information becomes available. This is unavoidable. Additionally, the advantage of vertical/horizontal scaling may change over time. When the number of processes is small during early (pilot) stage, it may make sense to maximize the speed of deployment and scale horizontally until clear groups of processes emerge. As the number of models grow, similar models can be ”merged” into a vertical model. Conversely, it may make sense for a vertically scaled model to be split up into separate horizontal models. If, for example, an adapted vertical model shows improved performance for some processes but worse performance for others, it may make sense to continue with separate horizontal models (essentially ”branching” the existing version). This is schematically depicted in figure 8.

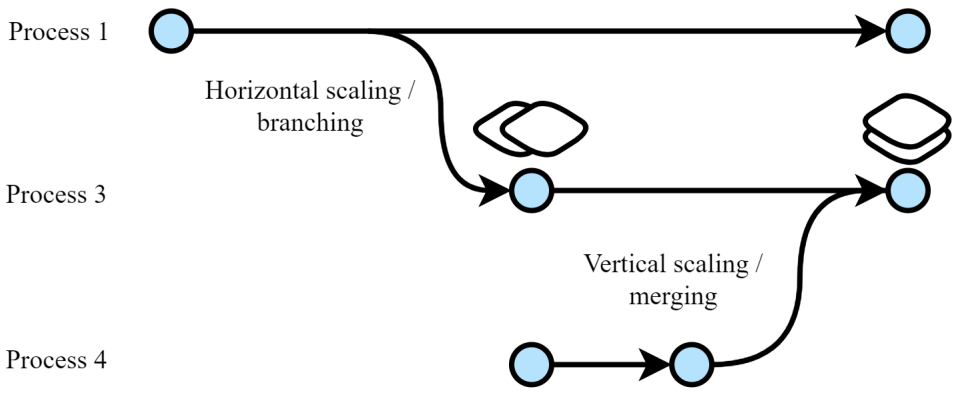


Figure 8: The advantages of scaling a model horizontally/vertically may change over time.

In summary, both horizontal and vertical scaling of ML models have relative advantages that may vary over time. As the number of processes and the available information changes, so too must the degree of vertical/horizontal scale. For the system as a whole to remain scalable, this adjustment must be automated. Thus, an automatable strategy for evaluating different scaling options is required to find the best path forward. This leads us to the concept of *scalability by extension* which is a more meaningful definition of scalability in the context of ML-systems:

Definition 4 *Scalability is the ability of a system to handle increased workloads by repeatedly applying a cost-effective strategy for extending its capacity [72].*

This definition of scalability considers the resources required for system maintenance and operation. The fact that this strategy must be cost-effective (in terms of required *resources*) is vital. Clearly, it does not make sense to scale an inoperable system.

2.1.2 Maintaining machine learning models

Once the first model is deployed into production, resources are required to operate and maintain the system. The major cost driver of a software system is not development but operation. This includes investigating and resolving failures, adapting the system to new infrastructure and repaying technical debt [36]. ML systems incur additional costs associated with model life cycle management (MLM). This section focuses on the challenge of maintaining ML models throughout their operational phase that extends from deployment of the model until the moment it is retired. Maintainability is the ability to restore a failed component to a specified condition [76]. In the context of ML models, this means restoring a degraded model to satisfactory performance. An ML model is said to be degraded, if its predictions no longer match the ground truth of annotated data with sufficient accuracy. This occurs due to a change in the input data that may occur gradually over time (concept drift) or abruptly (due to a so-called change point) [77]. This problem is highly relevant for ML models in production. Due to the large number of dynamic influences in a manufacturing environment, processes may change frequently. A real-world example is the cyclical change of the torque curve over the year as the elastic properties of the fastener change with the ambient temperature on the shopfloor. When the input data deviates too far from the training data of the ML model, it may no longer be able to monitor the process effectively.

To maintain ML models in production requires the ability to (1) detect and ideally anticipate model degradation and (2) retrain and potentially debug degraded models. The simplest way to detect model degradation is an apply-to-failure approach, where the model is kept in production until feedback from SMEs raise the issue to the data scientist. This may erode user confidence in the system. Alternatively, the input data can be monitored for concept drift and change points to anticipate model degradation. Once a degraded model is detected, the model must be retrained. The standard approach is to optimize the hyperparameters of the model to restore its performance. This may be difficult to automate for imbalanced data. If this does not work, data scientists must resort to manually debugging the model. Essentially, they try to retrace what the model has learned to understand why it fails to classify objects correctly.

This can be very challenging and time-consuming. The pattern detection and decision making processes of ML models is usually complex and difficult to interpret. An interpretable model would facilitate debugging. We consider a model interpretable, if the reason for its decision can be understood. This would allow data scientists to ”look inside the model” and more quickly identify the cause behind the low model performance. Additionally, the exchange of information with SMEs is an indispensable part of system operation. In practice, an unreliable model quickly erodes the confidence of domain experts [78, 79, 80]. This is made significantly worse if it is difficult to understand the reasoning behind a model decision. The ability to explain the model's decisions (even if these decision are incorrect from an the perspective of an SME) helps restore this confidence. These motivations are often cited in literature [81, 82, 83] and have lead to an increase in research aimed at improving the interpretability of machine learning models [84, 85].

2.2 Anomaly detection

2.2.1 One-class classification

The goal is to build a monitoring system that is able to detect patterns in the process data that indicate quality faults. The most straight-forward approach would be to implement a set of rules that can be used to detect such patterns. This rule-based classification of process data is how conventional process control system works. However, this is approach is not feasible for *unknown* patterns. The system would have to rely on rules to detect *possible* patterns defined by SMEs based on educated guesses about possible faults and how they would be discernible in the data. This approach is bound to fail - after all, if it was known with certainty what could fail and how it can be detected, there would be measures in place to do so. Even the most thorough failure mode and effects analysis (FMEA) performed by engineers cannot account for many of the faults that ultimately lead to product recalls. Additionally, production systems are subject to numerous dynamic influences that may lead to the onset of new fault mechanisms over time. Even in the (rare) cases where concrete data is available, SMEs tend to find it very difficult to define suitable classification rules. While it is often possible to judge if a given pattern may indicate a fault, it is much more difficult to define a set of rules by which such patterns can be detected that generalize well to large amounts of data. This I-know-it-when-I-see-it mentality is common in practice. Additionally, as is common for real-world problems, the boundaries of what constitutes a fault are often fuzzy. There is the fundamental problem

with a purely knowledge-based monitoring system. Instead, the desired system should detect the ”known unknowns” - patterns that SME know they are missing with their conventional process monitoring system without knowing what they look like.

In order to detect faults preemptively, a system is required that detects patterns that *may* indicate an underlying fault. In principle, patterns that do not conform to the expected normal behavior of the underlying process may indicate a fault. These non-conforming patterns are called anomalies. This does not mean that every anomaly *is* a fault, merely that all faults that *can* be detected will be anomalies. This important distinction is visually depicted in figure 9. Instances in category II, where a fault is present but no anomalous pattern exists in the data cannot be detected by monitoring the process data. The anomalies that do exist may or may not indicate a fault (category I and IV, respectively). Note, that the transition from normal to anomalous patterns is often fluid. Defining a suitable threshold is difficult and often based on arbitrary assumptions (this is known as the *threshold problem*).

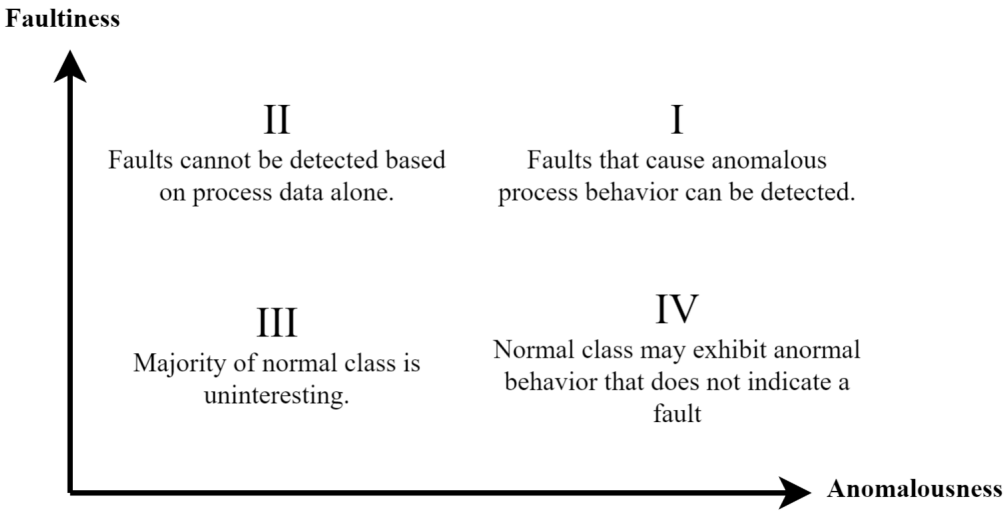


Figure 9: Schematic grouping of the anomalous and faulty objects that may or may not coincide.

Anomaly detection is relevant for countless applications in virtually every domain. It may come as somewhat of a surprise, therefore, that the term anomaly is used very inconsistently in literature. This is largely attributable to the ambiguous definition of what constitutes an anomaly.

Definition 5 *An anomaly is a pattern that does not conform to the expected behavior [86].*

Expected behavior may refer either to (1) recurrent patterns that are representative of the majority of the data or (2) patterns that a domain expert familiar with the underlying process would

expect to see for a fault-free process. Essentially, this is the difference between a data-centric versus a domain-centric perspective. Alternative names such as outliers, discords, novelties or exceptions are sometimes used in literature to try to alleviate this confusion. However, their definitions in literature are inconsistent and they are often used interchangeably [86]. To avoid having to differentiate between these vaguely defined terms and risk adding to the confusion, they are collectively referred to as anomalies in this work. In the context of this thesis, anomaly detection is the detection of non-conforming patterns from a data perspective. The goal, of course, is to ensure that these anomalies coincide with underlying faults as much as possible. This is a very challenging problem. An unsupervised ML model trained on the data will result in a naive anomaly detector that cannot make this distinction. Consequently, these models will either fail to reliably detect faults or suffer from a large number of false positives. Instead, the model should learn a comprehensive representation of the normal class that includes anomalies that *do not* indicate a fault. This approach is called *one-class classification*. Essentially, an object is an anomaly if it does not belong to the normal majority class.

2.2.2 Continuous learning based on user feedback

The decision of whether or not an object belongs to the majority class requires a decision boundary. However, real-world data is noisy and the boundary between what is normal and anomalous is often fuzzy. Objects of the fault class can be (subjectively) closer to the normal class than individual objects of the normal class to each other. For a real-world example from the manufacturing domain, refer to figure X in section Y. Additionally, the normal majority class may change gradually over time due to influences like tool wear or changes in the ambient environment. This is known as *concept drift* and leads to degradation of model performance over time. All of this results in a complex and changing decision boundary. This is schematically depicted in figure 10.

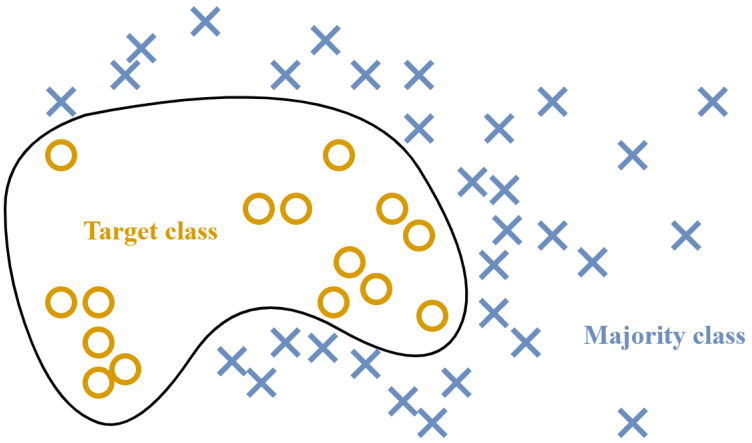


Figure 10: Objects of the target class may be (much) closer to objects of the majority class than to other objects within that class.

How can an ML model learn this complex decision boundary that is necessary to define the normal class? The simple answer is that it can't - at least not initially. The decision whether an anomaly is indicative of a fault or merely a rare event that has no effect on product quality can only be made by an SME. Therefore, SMEs must be integrated into the model training process so that these models may learn directly from user feedback. Since the data may change over time, keeping the models up to date requires a continuous cycle of user feedback and model retraining. This human-in-the-loop approach to training ML models has gained enormous popularity in recent years.

Figure 11 shows the number of relevant papers published in the International Conference on Machine Learning (ICML), International Conference on Learning Representations (ICLR) and the Conference on Neural Information Processing Systems (NeurIPS) over the past decade. This trend reflects a growing interest within the research community. The motivation is clear: rather than spending resources on fine-tuning models, focus on early model deployment and rapid improvement cycles.

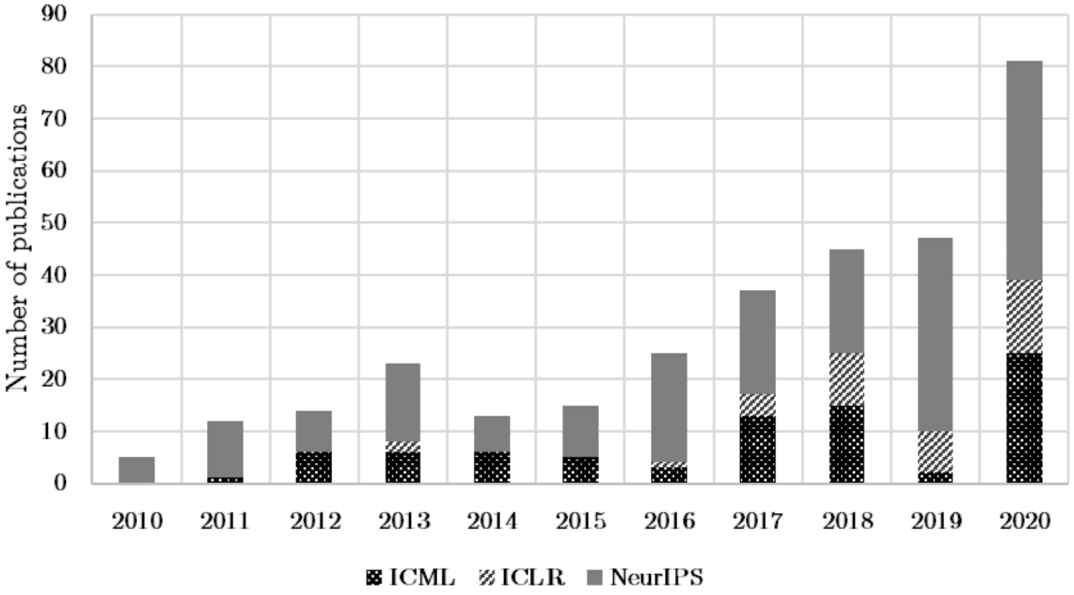


Figure 11: Number of papers published on the topic of adaptive learning in some of the highest-ranking venues for ML research from 2010-2020.

There are two elements that must be considered when setting up a continuous learning process. First, the learning cycle must be initiated with little prior knowledge about the data (in particular, few if any labeled data points are initially available). This is often referred to as the *"cold-start"* problem [87]. Effective model initialization is important to ensure an efficient feedback cycle. We can leverage the fact that the vast majority of the process data is representative of the normal class by using a semi-supervised learning strategy for model training. Second, the resources of SMEs are often extremely limited. This means, that the amount of required feedback to train the one-class models must be as low as possible. The system should query the user for labels on the most informative data points that allow it to efficiently retrain the model. This is known as *active learning*. The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns [88]. An active learning model poses queries in the form of unlabeled data instances to be labeled by a human annotator. Active learning is well-motivated in many modern machine learning problems, where unlabeled data is abundant and readily available but labels are difficult, time-consuming, or expensive to obtain [89]. Manufacturing data scientists need to face both of these problems when developing and operating an adaptive system for fault detection. Unsurprisingly, many researchers have started to address this issue by employing active learning strategies for anomaly detection [90].

2.2.3 Learning from rare events

The previous section explained (1) why anomaly detection is necessary for preventive fault detection, (2) why efforts should be directed towards continuous learning of one-class models based on user feedback and (3) how the active learning paradigm can be used to makes this feedback cycle more efficient. While the ability to train one-class models with little user feedback improves the practical feasibility of the system, it becomes absolutely necessary when trying to detect a specific fault class. Known faults that go undetected will quickly erode the confidence of SMEs in the system. In practice, it is very difficult to use the same one-class models for this classification task. The decision boundary is continuously evolving and cannot be guaranteed to reliably detect a specific class as the training data changes over time (for the reasons discussed in the previous section). Instead, it makes sense to use a dedicated classification model per fault class. The challenges of training a classifier on extremely imbalanced data using only a few labeled objects is well documented in literature. It is generally difficult to infer rules that generalize well to large amounts of new data from only a few samples. Although most real-world problems of interest are imbalanced [91], it is especially pronounced for fault detection in manufacturing: as soon as a fault is found, shop floor personnel will put measures in place to prevent it. Consequently, there will never be large amounts of annotated data for most faults. Their early detection (and remedy) is both the goal of the system and, at the same time, its curse from an ML perspective.

2.3 Pattern recognition in time series data

Time series data has become ubiquitous in virtually every domain [92] due to the increase in sensor connectivity and capability to process and store large amounts of data.

Definition 6 *A time series is a sequence of values* $T = t_1 \dots t_n$ *stored in an ordered list.*

The spacing of the observations in time need not be constant. Any sequence of values can be though of as a time series, irrespective of its temporal context. For example, handwriting [93], speech recordings [94] or DNA sequences [95] can be interpreted as time series. Even two-dimensional shapes can be projected to one-dimensional sequences [96]. In fact, this done quite regularly to take advantage of techniques for pattern recognition in time series data and apply them to image recognition applications. Analyzing time series data is known to be relatively challenging for two reasons. First, it requires analyzing a large number of correlated data

points. Time series databases may contain trillions of data points. ML algorithms have difficulties learning structures in high-dimensional data - this is known as the curse of dimensionality. Additionally, data points typically depend on preceding data points. Thus, observations are not independent and identically distributed random variables, which violates the assumption made by many ML algorithms. Second, an individual data point is usually only meaningful when viewed in its *temporal context* i.e. in relation to the values of its neighboring observations in time. Evaluating observations in isolation on a point-by-point basis is often meaningless. This is similar to the field of computer vision, where the color value of a single pixel in an image says nothing about its contents. This means, that effectively analyzing time series data requires the ability to recognize *patterns* in the time series. In practice and even most scientific papers, a precise definition of what constitutes a time series pattern is lacking, often resulting in unnecessarily abstract formulations. In a practical sense, a time series pattern is any (characteristic) subsequence of a longer time series. Depending on the problem, the length of a pattern can span multiple orders of magnitude, ranging from a small number of timesteps to the entire length of the sequence. The challenges of pattern recognition in time series are similar to those in image recognition. A model trained for object detection in image data should detect the object irrespective of its location, scaling, rotation, distortion, etc. Similarly, models for pattern recognition in time series data should be invariant to (1) spikes and dropouts, (2) the offset and phase shift of the time series and (3) distortions in amplitude and time of the pattern [97].

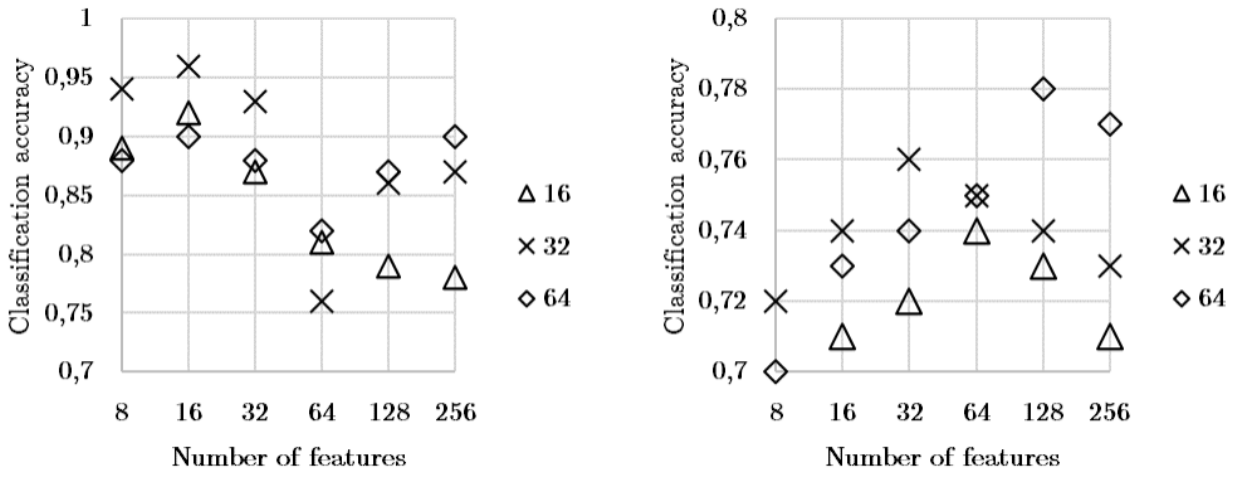
The task of pattern recognition can be approached in one of two different ways: template matching and feature analysis. There is some physiological evidence from neurological studies indicating that biological brains use similar mechanisms [98, 99]. Models based on the template matching approach classify a time series pattern by comparing it to a set of stored templates. Feature detection models rely on the extraction of higher-level features that describe interesting properties of the data. Both of these have their merits and are widely used in computer vision and time series analysis. The main challenge of both approaches is automating the extraction of effective templates or features from the data. The extraction process is accompanied by an unavoidable trade-off between a loss of information and the number of templates/features that are extracted. More features generally results in a lower information loss but increases the dimensionality of the feature space, making the problem increasingly intractable for conventional ML algorithms. As discussed in section 2.2, this template/feature extraction process

must work with both a small amount of annotated data and entirely unsupervised. In recent years, advanced data mining (DM) algorithms to extract descriptive templates from large sets of time series data have been developed and are quickly gaining popularity for many practical applications [100, 101, 102]. These DM algorithms will be discussed in section 4.2. By comparison, many of the widely used feature extraction techniques for time series data were developed with the intent of data *compression* rather than description/classification. Examples include the Fourier or wavelet transformation (these have been shown to be similarly effective depending on the application [103]), piece-wise aggregate approximation (PAA) such as the SAX-notation [92, 104] or adaptive piece-wise constant approximation (APCA). While many generic and purpose-built feature extraction algorithms have been published over the years [105], they are used much less frequently. Finding a trade-off between data compression and loss of information is extremely challenging, especially in an unsupervised setting. To get a sense of this problem, consider the classification results of a 1-NN classifier trained for different SAX feature representations of real-world manufacturing data. The 1-NN classifier is typically chosen as a benchmark classifier due to its robustness and competitive performance for many real-world problems [106, 107]. The data is describes in detail in section 6.1.2. The SAX representation was chosen because it allows a systematic grid search of both the number of features and the number of values these features can take. Both were varied logarithmically across a wide range. The results in figure 12 show, that different fault patterns require vastly different feature representations to be accurately classified. The task of finding a problem-specific trade-off cannot be automated for unlabeled data [108] and is extremely difficult for imbalanced data - there simply is no way around this problem.

2.4 Summary of requirements

The challenges and requirements discussed in the previous sections are recapitulated below. Their relationship is graphically depicted in figure 13.

- A knowledge-based monitoring system will reflect only past knowledge and is unsuited for detecting unknown patterns. It is these unknown patterns that we wish to detect. Preventing faults requires an anomaly detection system.
- The majority of the data is unlabeled and belongs to the normal class. Anomalies can be detected via a one-class model of this normal class.



(a) Class 1 (b) Class 2

Figure 12: Accuracy of a 1-NN classifiers trained on different SAX feature representations of real-world manufacturing data.

- The large number of uncontrolled influences in the manufacturing environment result in a complex decision boundary between the normal and target class that evolves over time. Thus, the one-class model must learn continuously from user feedback on its classification results.
- Process data is predominantly unlabeled. Learning from this data requires semi-supervised learning strategies to maximally leverage the annotated data that is available.
- To monitor a large number of processes requires the ability to efficiently train and adapt a large number of ML models. Model management must be highly automated and orchestrated.
- Process data is typically time series data. This requires ML models that are able to recognize time series patterns.

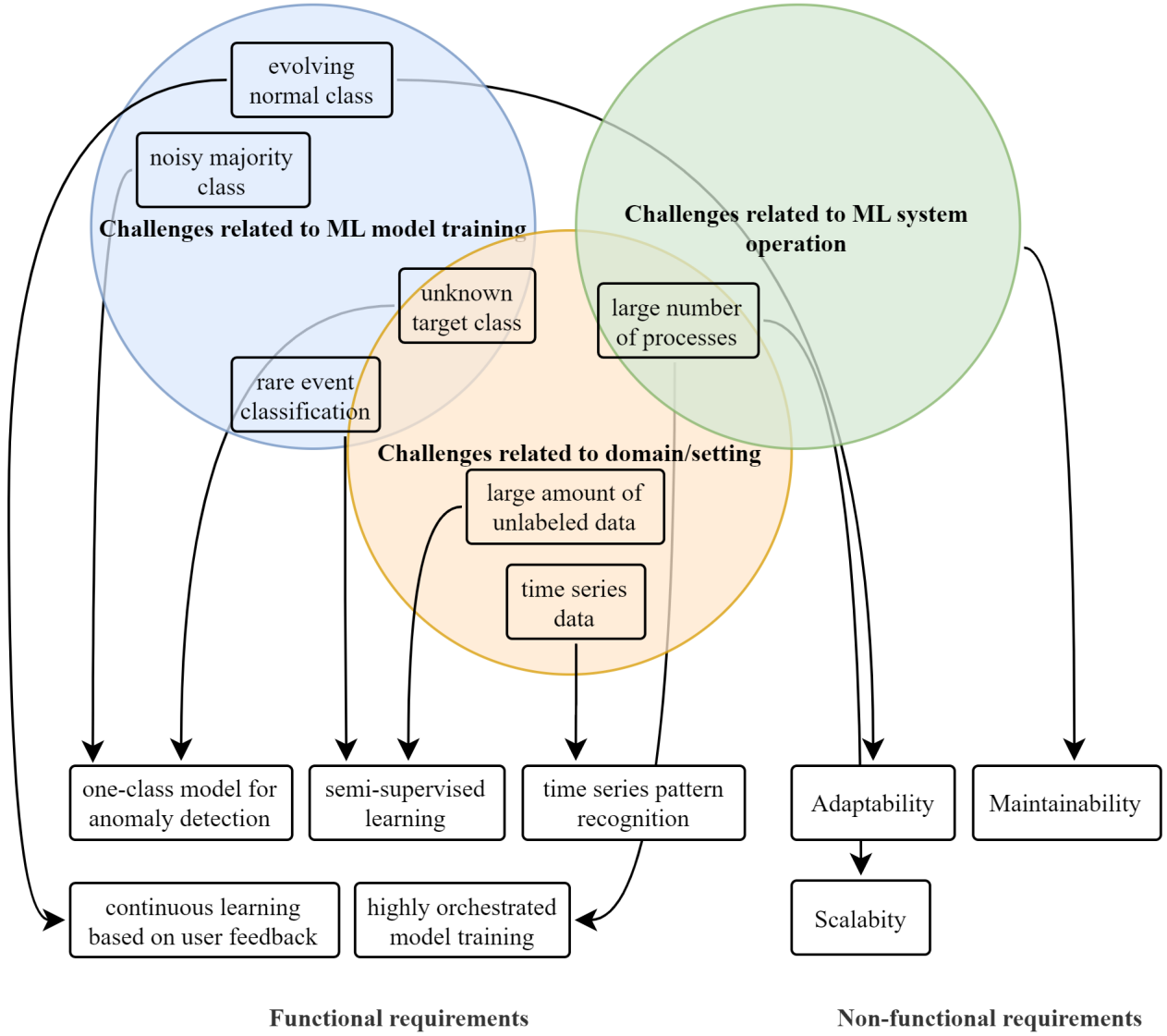


Figure 13: Challenges and requirements for an ML system for anomaly detection in manufacturing.

3 State-of-the-art in research and practice

There is a vast amount of literature on ML applications for anomaly detection. These range from the application of out-of-the-box ML algorithms to the development of new algorithms and from proof-of-concept testing on public benchmark datasets to large-scale ML systems. To provide an overview of this literature and determine the state-of-the-art that is relevant to the scope of this thesis, the papers were categorized according to the requirements identified in the previous section. While most papers consider only a subset of these requirements, two important requirements for large-scale industrial ML systems were identified as particularly underrepresented, namely (1) the adaptability and (2) the scale of the (productive) ML sys-

tem. In an effort to base the evaluation on objective criteria and eliminate personal bias, a set of criteria was used to judge each publication (cf. table 2). Each criteria was evaluated qualitatively on a range from 1 to 5 and the values added up to get a final score.

Dimension	Criterion
Adaptability	Unsupervised training to overcome cold-start problem
	Ability to learn from rare events & extreme class skew
	Interactive learning cycle can be automated
	Field-tested in real-world setting with SMEs
Scalability	Orchestrated model training process
	System monitoring and quality control measures
	Field-tested under operating conditions at scale
	Documentation of system architecture and workflow

Table 2: Questions used to evaluate publications regarding their ability to meet the requirements in section 2.4.

Based on this evaluation, the literature can be grouped into four approximate groups as shown in figure 14. By far the biggest group (B) is comprised of publications that report the results of out-of-the-box applications of existing ML algorithms or propose novel changes to those algorithms for niche anomaly detection applications. Most of the investigations include benchmark studies and some (often limited) field-testing in real-world settings. However, there is no focus on large-scale deployment and operation of a productive system. Where no information is available that indicates otherwise, it is assumed that these systems have not made it past the pilot stage. The majority of these publications are focused on anomaly detection in time series in the manufacturing domain - generic publications on the subject or those from other domains were only included if it contained additional information relevant to the requirements in section 2.4.

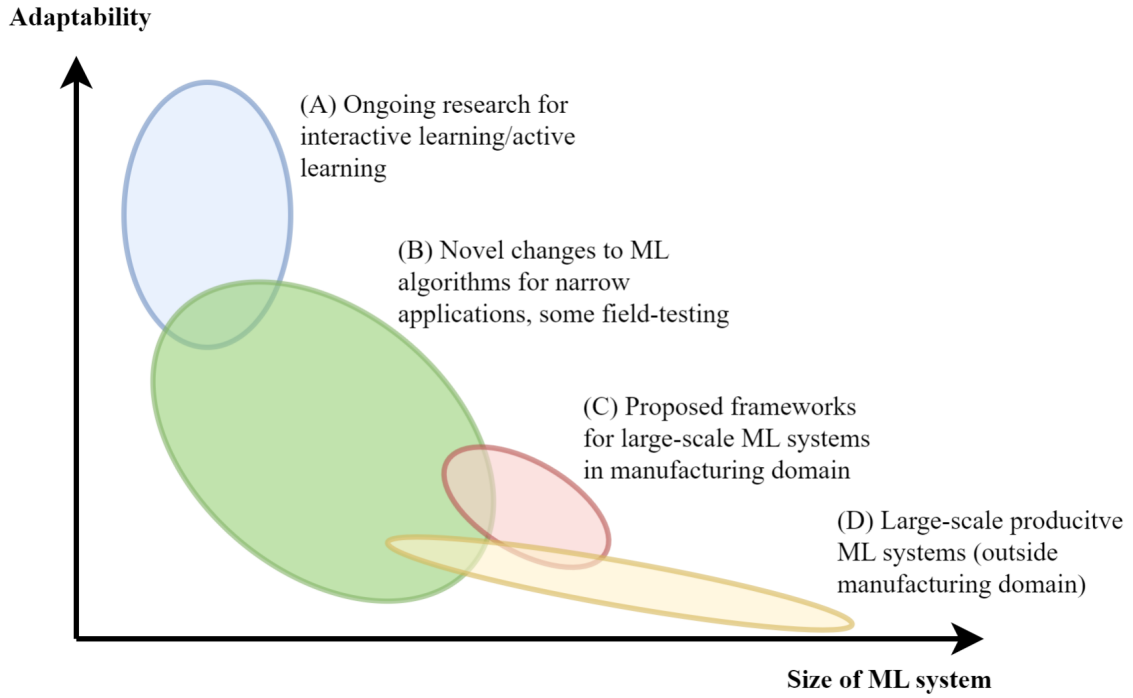


Figure 14: Clustering of literature on anomaly detection systems according to their adaptability and scale.

3.1 ML algorithms for anomaly detection

The simplest approach from an ML-perspective is to reduce the dimensionality of the time series data and use ”out-of-the-box” ML algorithms to detect anomalies in this lower-dimensional space. The quality of this approach fundamentally depends on the effectiveness of the dimensionality reduction technique. The difficulty of doing this in an unsupervised setting has been demonstrated in section 2.2. An alternative to this approach is the extraction of descriptive features from the time series. The goal is to capture the characteristic properties of the time series in a (much smaller) set of features. Typically, the time series is segmented using a sliding window from which statistical features are extracted [109, 29]. Anomalies are detected by applying unsupervised ML algorithms (e.g. one-class support vector machines (OCSVM) [110], local outlier factor [29]) on the resulting feature representations. This approach has been used for anomaly detection in various manufacturing processes e.g. metal forming [28], steel rolling [109] and semi-conductor manufacturing [30, 29].

The frequency of publications based on this approach have been declining over the years due to the inherent difficulties associated with extracting meaningful features. There is no generic one-size-fits-all approach to minimizing both the number of features and the loss in information

is not possible. This was demonstrated to be the case in the introductory section 2.3. For unsupervised settings, finding a trade-off requires profound understanding of both the data and the feature extraction algorithm - at which point it is often easier to define these features manually. Consequently, researchers have increasingly turned to other approaches, particularly the use of deep learning. Numerous benchmark studies attest a superior performance of DL models for a broad range of anomaly detection applications [111]. Frequently cited advantages of DL models are the ability to deal with variable-length input data [110] and to capture complex dependencies between timesteps [28]. Essentially, they are credited with being both easy to apply (thanks to the high-level nature of open-sourced ML frameworks) and powerful (complex features can be learned from the data without requiring manual input). Standard DL models have been piloted for anomaly detection in the manufacturing domain [26, 112] and other engineering fields like power sensors for smart grid monitoring [113] and telemetry data [114] with some success. An often criticized drawback is the lack of interpretability of DL models [84, 85] and the associated difficulty of systematically improving their accuracy. Due to the large number of model parameters, DL models are often treated as black-box models where data scientists rely on (more or less well-founded) hyperparameter-tuning to reach sufficient model accuracy. While ensemble DL models may improve accuracy on some (benchmark) datasets [115], it changes model interpretability from bad to worse. Some researchers have tried to address this by making the model itself more interpretable, like simplifying the topology [116] or formulating a dedicated learning objective [117]. Others have built software systems that allow to visually explore DL models via a graphical user interface [118, 119]. This process of hyperparameter-tuning is often tedious, especially when maintaining a large number of models in production. Whiel there are experience reports from the industry that purportedly show the use of these models at scale, little to no information is provided about the (long-term) feasibility of these systems [120]. To date, no detailed experience report exist that focuses on the large-scale implementation of unsupervised/semi-supervised DL models for anomaly detection in time series data.

Following this initial wave of applying DL models as a black-box end-to-end approach for anomaly detection, researchers have begun to leverage the models' feature-learning ability. DL models are used as feature extractors and combined with simpler and more interpretable ML classifiers. Essentially, this is a return to the two-step approach of feature extraction and classification discussed in the beginning. This approach offers competitive accuracy on benchmark

datasets [110] and has been integrated into stand-alone algorithms for anomaly detection applications [121, 122]. More importantly, the model output allows for better interpretation of what the model has learned. In some cases, the model's feature representation of the (unlabeled) input data can be used to explore underlying structures in the data. To do this, the latent variables of the unsupervised model are summarized using dimensionality reduction techniques [123] and clustering algorithms applied to the final feature representation [124, 125]. Some researchers have used generative models [126] with dedicated learning objectives [127] to help interpret the latent space. While there is much to be said of this approach, unsupervised feature learning using DL models is challenging and has become a fringe area of research in its own right [128, 129]. Aside from data exploration, the learned features can be used to detect anomalies directly (e.g. for monitoring of ECG measurements [130]), including one-class models for anomaly detection [131, 132].

3.2 Retraining models in production

Changes in the data (or understanding thereof) are common in practice, which makes the issue of model adaptability central for productive ML systems. Data scientists are often concerned with ways to reliably detect concept drifts in the data and retrain the model to avoid model degradation. For anomaly detection in particular, the primary focus lies on detecting changes in the normal class of the data [133]. A number of suitable methods have been proposed and verified for both benchmark and real-world data [134, 135]. For productive systems, mechanisms for concept drift and change-point detection are used to automatically trigger the retraining process [136]. This is a basic, yet important tool for quality assurance of productive ML systems. The second aspect, namely the ability to retrain models based on evolving *understanding* of the data, has enjoyed a growing increase in research interest in recent years. As discussed in section 2.2, active learning promises to resolve this problem. The ability to continuously adapt models based on human feedback makes it possible to deploy models into production faster and quickly close the training/serving-skew through iterative improvements. This avoids the up-front cost of labeling large datasets that is often prohibitively expensive in practice [137]. Researchers agree, that the potential of this human-in-the-loop machine learning is an important but often neglected potential for real-world ML systems [138]. Although its feasibility has been demonstrated for a range of benchmark time series datasets [138], existing research remains mostly experimental with limited field-testing. Research focus has been devoted to finding efficient labeling strategies for settings where the classes are extremely imbalanced and

random sampling would subsequently lead to a lot of wasted effort [20]. Most approaches for active learning of anomaly detection models use an initially unsupervised model to give the SME recommendations on which objects to label. The model is retrained in an interactive data exploration loop to make the anomalies more relevant to the user [90]. This has been demonstrated for anomaly detection in time series data [90] and object recognition in image data [88]. These interactive human-in-the-loop learning cycles require extended interaction between the user and the system.

3.3 Productive large-scale ML systems

The practical aspects of ML system operation (especially at scale) have only started becoming a focus of research in the past decade. Due to the limited amount of literature, it is important to look beyond the manufacturing domain and time series data. From the titles alone, one would be forgiven for wrongly assuming that a framework for large-scale operation of productive ML systems for anomaly detection in time series data already exists. For example, consider the following titles of published manuscripts: ”generic and scalable framework for automated anomaly detection on large scale time-series data” [27], ”framework for large-scale process monitoring” [139] or ”framework for end-to-end deep learning-based anomaly detection” [140]. Despite their assertion of having solved this problem, these papers lack any definition of what constitutes scalability, let alone a metric to evaluate it. The published frameworks are little more than a proof-of-concept of a marginally new ML algorithm tested on a hand full of benchmark datasets. They fail to address almost all of the challenges discussed in section 2 and are only marginally reproducible at best (a point that will be discussed in more detail at the end of this section). Li et al. propose an automated approach to training an end-to-end anomaly detection pipeline [141]. The approach aims to automate both algorithm selection and hyperparameter optimization (HPO). This is known as the combined algorithm and hyperparameter selection (CASH) problem, which is notoriously difficult to automate for unlabeled data. Additionally, it relies on an effective feature extraction subroutine with all the associated difficulties. Chen et al. propose a modeling framework to train a hierarchical Bayesian network for large-scale process monitoring [139]. The framework was evaluated on a single (synthetic) benchmark dataset of chemical processes that is comprised of 40 variables from six processes. This is hardly ”large-scale” - as discussed in section 1, the number of processes in a production systems can easily be two orders of magnitude higher. Stojanovic et al. propose a ML system for process monitoring that is much simpler, although tested on even fewer processes [142]. Laptev et al. published a

report on a framework used at Yahoo that uses a collection of anomaly detection and forecasting models in a layered structure to detect anomalies in the time series data [27]. The framework was validated on both synthetic and real-world data. A key aspect of the framework is its modular structure, although the design of these modules is not explained. Following a similar trail of thought, Ren et al. proposed a modular framework for preprocessing, analyzing and classifying time series [143]. The system is used for various (unspecified) time series anomaly detection applications at Microsoft. All of these approaches exhibit the following critical shortcomings:

- The description of the system architecture is limited to a high-level discussion (if presented at all). Details on model management are not provided. Thus, reproducing the system as described in the publication is not possible.
- Experience reports (let alone quantifiable metrics) on the maintainability of the ML models and the system as a whole are not provided in any publication.
- The challenges of adapting models based on feedback from SMEs is not addressed in any publication. Although the importance of adapting models is explicitly stated [142], no solution is proposed.

These publications from the manufacturing domain are ill-suited as a starting point for further research. Instead, they merely serve as evidence of an increasing shift in research focus towards bringing anomaly detection systems into production.

Despite this sobering verdict, there are some (few) examples of applied research (outside the manufacturing domain) that report on many of the same challenges discussed in 2. These offer interesting insights into possible design considerations for the framework and the ML algorithms. Sculley et al. published as paper on a large-scale ML system for adversarial advertising (e.g. detecting phishing, malware or counterfeit goods) at Google consisting of thousands of models [41]. Although the system monitors text rather than time series data, the issues of multiple unknown classes and a large majority normal class are similar, including all the associated challenges of learning from these extremely rare events. The system uses conventional ML algorithms (nearest neighbor, naive bayes and linear SVM classifiers), focusing on minimizing system complexity. The need for effective system monitoring and a high degree of automation are highlighted. Additionally, the need and potential of active learning is reaffirmed, making it possible to rapidly develop and deploy new models, often requiring only a few dozen hand-labeled examples. Raeder et al. report on a similarly large system in operation for display

advertising [144]. Here, too, a fundamental focus is placed on the ability to learn from the data despite extreme class skew (up to 1 in 100,000 objects belong to the target class). The major focus of the research was to ensure the ability to scale and extend the system by automatically adding new models to ”minimize human intervention” (cf. the discussion of scalability by extension in section 2.1.1). Both systems are much more advanced, both in terms of scale as well as degree of automation, than those reported in the manufacturing domain. The authors focus on many of the same key challenges that have been discussed in the previous section. Specifically, the need to integrate an active learning procedure at scale [41], learn from rare events, and designing both the algorithms and the system from the ground up to cope with the challenges of maintaining a large number of models in production [144]. Interestingly, while DL models frequently outperform all other models for anomaly detection, both systems are ”large-scale data mining systems” [41]. This suggests, that the use of time series data mining algorithms is a promising alternative to deep learning models. However, both systems do not deal with time series analysis, which require disproportionately more sophisticated data mining approaches. In recent years, the research group around Keogh et al. has developed a number of DM algorithms for extracting patterns from time series data [106, 145, 146, 147]. These algorithms have already been used for anomaly detection in time series [148, 149] and been integrated into expert systems for interactive data exploration and pattern extraction [150, 8].

3.4 Reproducibility of results

To conclude this literature review, a comment on the reproducibility of published results is necessary. The scientific process is built on the idea of extending current knowledge by building on published results. However, the ability to reproduce the published results of applied ML research is often poor. This is because researchers are often not required or able to share proprietary data and source code. While the use of benchmark datasets alleviates the former problem, the results often do not translate to the real world. Meta-analysis of literature shows that the vast majority of results cannot be reproduced. This may be due to (unknowningly) cherry-picking the benchmark datasets or extensive hyperparameter optimization (HPO). An analysis of the reproducibility of the publications considered in this review was performed that considered the availability of the source code, whether the results were verified on benchmark data sets at all and if the paper offered at least some discussion of the hyperparameters. The results are shown in figure 15. While the trend of providing source code has increased over the years, it remains low, especially in applied research outside the computer science domain.

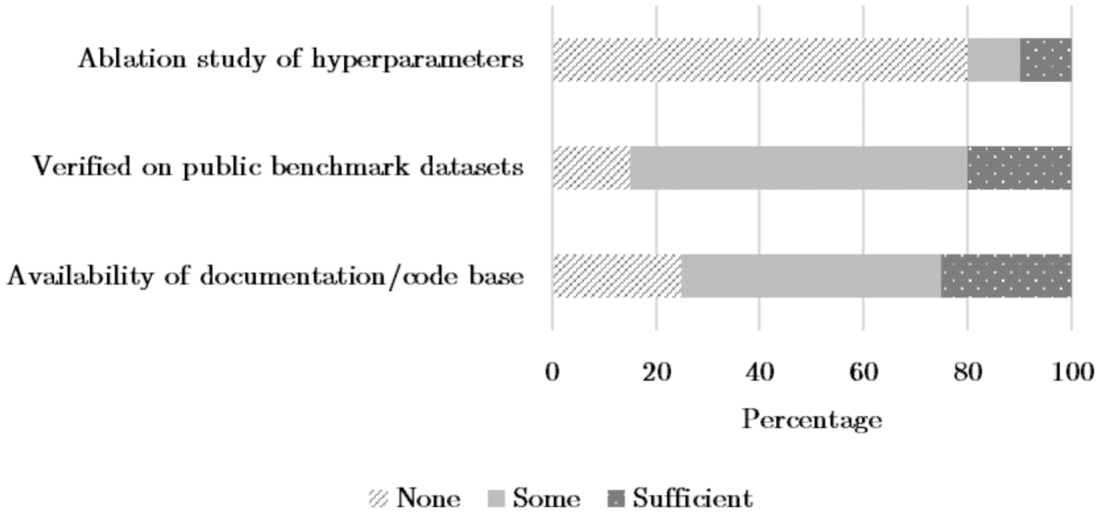


Figure 15: Qualitative evaluation of the reproducibility of available literature

Authors are often quick to equate marginal improvements over a number of prexisting methods on benchmark datasets with generalizable controbutions that easily translate to real-world improvements. Statements such as ”the proposed approach outperforms classical feature extraction procedures” [124] when only two other methods were tested seem like quite a leap. Other claims like ”our technique consistently improves all known algorithms by a wide margin” [132] seem at least questionable. More often than not, authors consider their algorithms to be scalable, robust or adaptable, without providing any evidence that supports their claims. Claiming that an algorithms ”automatically adapts itself to the anomaly detection use-case that is important to the user” [27] without providing a detailed explanation what this means and how this is done in practice contributes little to the current state of the art. This trend has already been discussed in section 1.3.1 and was used as a justification for employing the design science research methodology. To avoid these pitfalls, this thesis will not make the claim of a one-size-fits-all ML system for anomaly detection. Instead, the goal is a framework to develop such systems. The design considerations behind each research artifact will be explained in detail and verified on both benchmark and real-world data to allow third parties to adapt the framework to their setting. The source code and technical documentations for each artifact is made publicly available. ”Operating a [...] system at scale requires more than [...] brilliant algorithmic design. The consistent need to grow and change the system while maintaining or improving performance means that quality control becomes increasingly important as the system matures.” [144]. Therefore, an entire section is dedicated to the design of the ML system itself, that is used to evaluate both DL and DM approaches at scale under real-world conditions.

4 Time series data mining for anomaly detection

The discussion in the previous section showed, that the few productive anomaly detection systems operated at scale are essentially ”large-scale data mining systems” [41]. The first research cycle, therefore, built upon these insights by using State-of-the-art DM algorithms to develop a DM system for anomaly detection in time series data. These algorithms classify a time series by matching it with a library of time series patterns. While the *matching* operation itself is simple, the process of *extracting* these patterns and *classifying* objects requires more sophisticated DM algorithms. When applying these algorithms at scale to construct DM models, data scientists are faced with the same challenges discussed in section 2. The first subsection recapitulates the different motivations for using DM models for anomaly detection. The second subsection summarizes the fundamentals of time series pattern matching and state-of-the-art algorithms for supervised and unsupervised pattern extraction. The third subsection presents an approach that uses these algorithms to construct an unsupervised anomaly detection model. The fourth subsection presents an improvement of the state-of-the-art pattern extraction algorithm that improves the ability to extract effective classification patterns from highly imbalanced data. The last subsection shows how both contributions can be integrated into a single framework to train and adapt one-class models for anomaly detection based on continuous user feedback.

4.1 Motivation for pattern matching models

A pattern matching model uses a set of patterns to classify an object. The patterns are extracted from the data using DM algorithms and organized into libraries. These patterns can then be used as templates for time series analysis. Note, that there is a distinction between a template and a prototype. A template is based on a real instance observed in the past that is used to find a one-to-one match. A prototype, on the other hand, is a sort of average representation derived from shared features of various instances of the same class. The pattern extraction algorithms discussed in this thesis are template extraction algorithms. The model matches these templates to a time series object using a pattern matching subroutine. The result is a pattern-based feature representation of the object. ML models can then be used to learn decision boundaries for classifying the objects. This is schematically shown in figure 16. Therefore, a pattern matching model relies on both DM and ML algorithms.

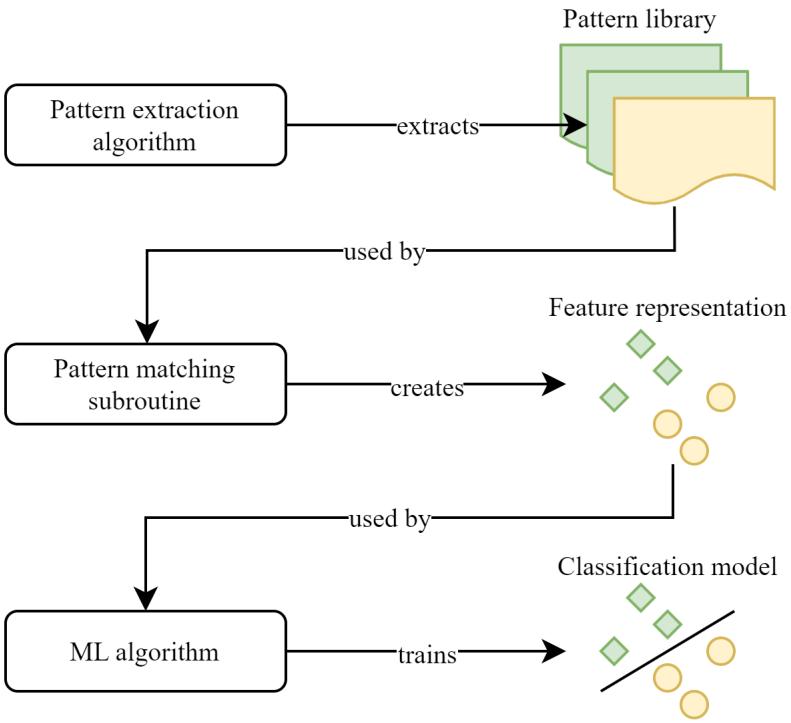


Figure 16: Schematic depiction of the components of a pattern matching model.

There are three factors that motivated the development of pattern matching models. First, these models can be trained so that their complexity matches that of the data. As discussed in section 2, overly complex models may overburden developers in practice. Ideally, the same ML algorithm will train simple models when possible and complex models when necessary. Pattern matching models allow this flexibility by simply extending or altering the library of patterns. If a target pattern is easy to detect, a simple model based on a small number of sequences can do the job. For more complex classes, the model can learn to classify an object based on a combination of the presence and/or absence of numerous patterns allowing for an almost arbitrarily complex decision process.

Second, this approach makes it possible to organize the knowledge learned by the model into a taxonomy that can be easily understood by humans. All information used by the model for pattern detection is contained in the sequences stored by the model. These can be categorized into groups and deviations of patterns that the model looks for. This helps developers and SMEs better understand both the model as well as the underlying data. Additionally, classifying an objects by matching it to a library of sequence templates/prototypes is a low-level method of pattern detection, resulting in models whose decision making remains relatively easy to interpret even for complex patterns. This greatly simplifies model debugging.

Third, this approach to pattern recognition is widespread in computer vision and shares similarities to the way biological brains work. Template and prototype matching are two basic theories in the field of neuroscience that explain the recognition of patterns as a cognitive process of matching perceived information with information retrieved from memory [98]. While this is a weak justification on its own for choosing a modeling approach, empirical observation of SMEs suggests that this process is indeed used when manually classifying data. When asked to explain what constitutes a particular fault class, SMEs refer either to the presence of a sequence that is not present in normal objects or the absence of a sequence that is always present. If they are unsure, they typically review a series of known normal objects (ground truth) before labeling the target object. This suggests, that the classification performed by SMEs is based on matching a set of normal sequences that have been committed to memory with the target object. Thus, the approach is essentially emulating the decision process it is trying to automate.

4.2 Fundamentals of pattern extraction and matching

This subsection summarizes the theoretical fundamentals of pattern extraction and pattern matching in time series data. The first half covers the low-level concepts and algorithmic subroutines used for matching time series patterns. The second half summarizes state-of-the-art high-level data mining algorithms used for unsupervised and supervised pattern extraction from time series data. These algorithms are the result of decades of research conducted by the research group around Eamonn Keogh [151, 152, 101, 153, 5, 145] that was identified as promising in section 3.

4.2.1 Time series subsequence matching

As discussed in section 2.3 a pattern in a time series T is a *subsequence* of that time series.

Definition 7 *A subsequence* $T_{i,m} = t_i, t_{i+1} \ldots t_{i+m-1}$ *of time series* T *is a shorter sequence of length* m *starting from index* i *where* $1 \leq i \leq n - m + 1$.

A time series can be broken down into its constituent subsequences by sliding a window of size m across the time series T. This yields a subsequence set A. There are $n - m + 1$ subsequences in the set A. A separate set exists for each m where $1 < m < l - 1$.

Definition 8 *A subsequence set* $\mathrm{A} = \{T_{1,m}, T_{2,m} \ldots T_{n-m+1,m}\}$ *of a time series* T *is an ordered set of all possible subsequences of* T *of length* m.

The definition of a subsequence set is provided to make it easier for readers to understand the pattern matching subroutine. A time series is said to contain a pattern if that pattern is similar to any subsequence in the set A. This evaluation is based on a simple matching operation, whereby a single query subsequence Q is compared to all subsequences in A. The sequence of distances between Q and all subsequences in A is referred to as the distance profile D [97, 154].

Definition 9 *A distance profile* D *of a time series* T *is a vector of the distances between a given query subsequence* Q *and each member in the subsequences set* A *of* T. *[149]*

Figure 17 depicts the distance profile of three query subsequences Q and a time series T. The time series is an ECG signal of a normal heart beat. Most readers will be familiar with the general shape, making it easier to understand the concept of a distance profile D. The subsequences correspond to the three main polarization regimes of the heart muscles. Note, that both the query subsequence Q and each subsequence in A must be z-normalized before calculating the distance between each subsequence-pair. This is important for the distance to quantify the similarity between their *shapes*, irrespective of the amplitude of the signal [4, 155].

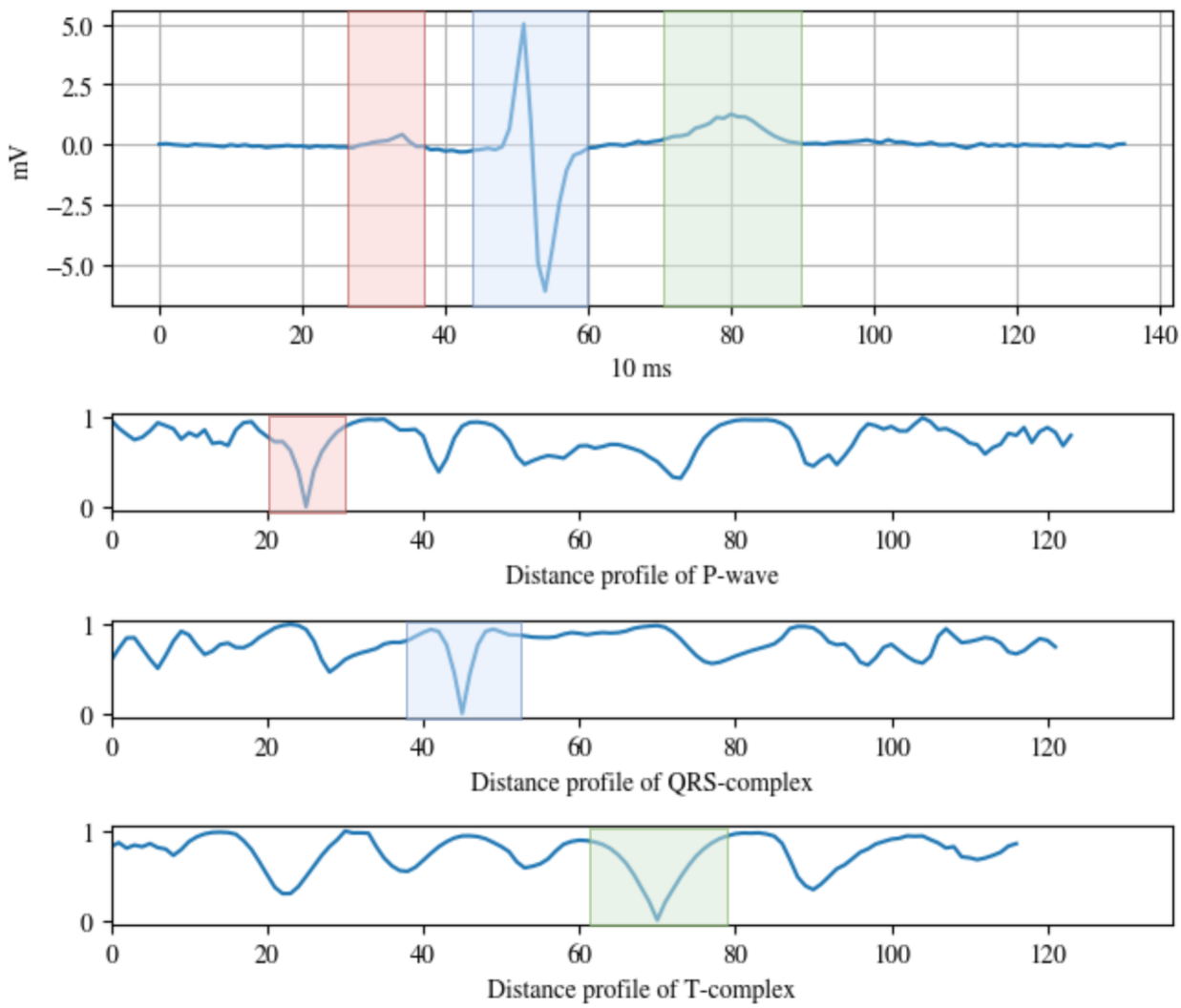


Figure 17: The (Euclidean) distance profile D of different query subsequences Q and a time series T.

Low values in the profile correspond to subsequences of the time series T that are similar to the query Q. The minimum value of D quantifies the *presence* of Q in T and corresponds to its nearest neighbor in the set A.

Definition 10 *The 1NN-function* $f(Q, \mathrm{A})$ *is a Boolean function, that returns "True" if* $\mathrm{A}[i]$ *is the nearest neighbor of* Q *in the set* A *and "False" otherwise. [149]*

The distance profile and the 1NN-function are two fundamental concepts in time series data mining. A vast number of high-level pattern detection algorithms in literature are constructed from just these two building blocks [156, 149, 102, 157, 147, 158]. By applying the 1NN-function to two all-subsequences sets A and B, the nearest neighbor to each element in set A among all elements in set B can be found. This entails comparing every element in set A with every element in set B and is called a time series similarity join [149].

Definition 11 *A similarity join* $\mathrm{J_{AB}}$ *between two all-subsequence sets* A *and* B *is a set of subsequence-pairs of each subsequence in set* A *and its nearest neighbor in set* B. *[159]*

Note, that some subsequences in B may be the nearest neighbor of multiple subsequences in A while others may not be the nearest neighbor to any subsequence. In most cases $J_{AB} = J_{BA}$. The similarity join is standard database operator used extensively for data processing in diverse application domains [159]. The efficient computation of set similarity joins has received much attention from both academia and industry [160]. A matrix profile can be defined based on the time series similarity join.

Definition 12 *A matrix profile* $\mathrm{P_{AB}}$ *is an array in which the Euclidean distance between each pair in* $\mathrm{J_{AB}}$ *is stored. [149]*

4.2.2 Distance measures

The discussion so far has focused on comparing the *similarity* between subsequences without considering *how* to measure this similarity. Generally, the only way to compare two objects is to calculate a distance between them, which is why the terms *similarity* and *distance* are often used interchangeably. Since distance values are typically unbounded (i.e. there is no upper limit a distance value can take), they are difficult to interpret in absolute terms [149]. In fact, depending on the distance measure in question, these values may not even correspond to what an SME would consider as similar in *relative* terms. Consider the real-world example of tightening data depicted in figure 18. A process expert would be hard pressed to say that ”object A is twice as similar to object B as to object C” and may even have trouble deciding which object is more similar at all. Either object can be more similar depending on the distance measure used. This example illustrates the difficulty of measuring the similarity of two patterns and explains why a vast number of distance measures have been proposed to more effectively measure the similarity between two sequences for different applications. The two most widely used distance measures for time series data are the Euclidean distance and the Dynamic Time Warping (DTW) distance. Most other distance measures for time series proposed in literature can be considered variants of these two that address specific variations in time series data [156].

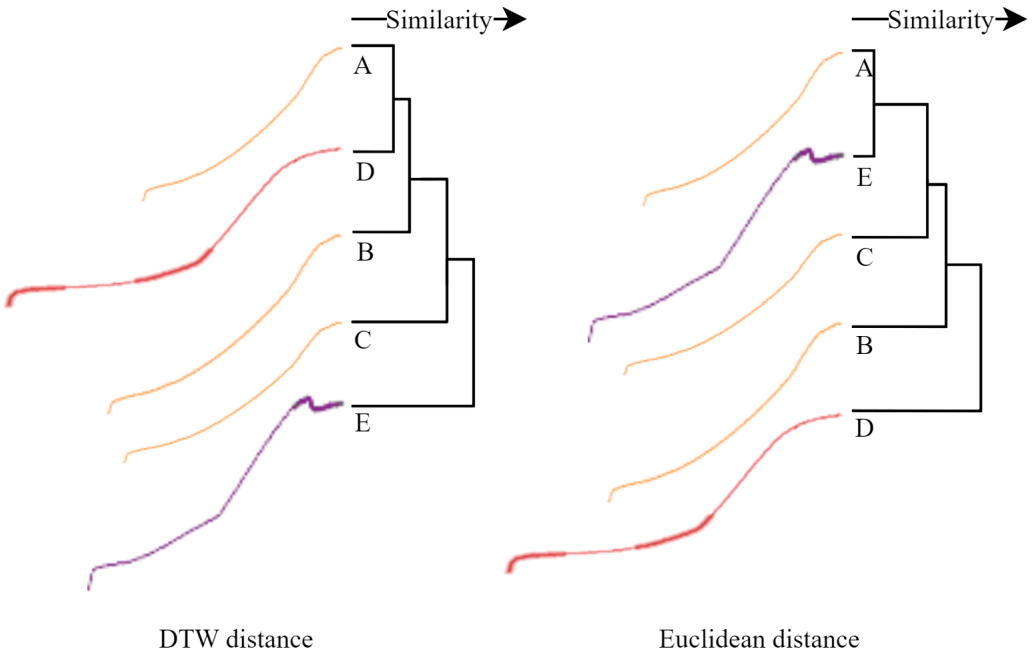


Figure 18: Similarity between real-world process data objects according to different distance measures.

Consider the simple case of two points in three-dimensional space, where each point is represented by its coordinates. The Euclidean distance is defined as the root of the sum of the squared differences between these vectors (this corresponds to the length of the shortest line connecting these points). The Euclidean distance calculation can be generalized to any n-dimensional space. Thus, the Euclidean distance can be calculated between two time series, provided they are of the same length. While simple, the Euclidean distance is often ill-suited for the comparison of time series data. Even small shifts or distortions of the underlying patterns may lead to a large Euclidean distances between two objects. This is referred to as (statistic) brittleness of the distance measure [156]. The Euclidean distance essentially considers every point along the time series an independent dimension, ignoring local structures of neighboring points. This is the same reason why it is difficult to find a meaningful nearest neighbor in high-dimensional space. Therefore, researchers developed elastic distance measures, the most popular of which is the DTW distance. This elasticity makes it possible to compare non-adjacent points of two time series. The DTW algorithm was developed in the 1970s to measure the similarity between two temporal audio signals for applications in speech recognition [161, 162]. Because people have different speaking rates, the speech patterns of the same spoken word may vary. DTW finds a non-linear mapping between two sequences by locally compressing and stretching the time axis of one sequence so that its signal is optimally aligned with the other. This local distortion

of the time axis is referred to as time warping and makes it possible to calculate a meaningful distance metric between both sequences. Figure 19 shows the mapping between two sequences as calculated by the DTW algorithm compared to the Euclidean distance for real-world data. This mapping allows to assess the similarity between sequences, even if their signals are misaligned or distorted. Two time series are considered similar if, after non-linear adjustment of the time axis, they can be made similar under Euclidean Distance [163]. This property makes DTW relevant for many sequence analysis problems, which has led to its widespread use for clustering [164], classification [74] and similarity search [165].

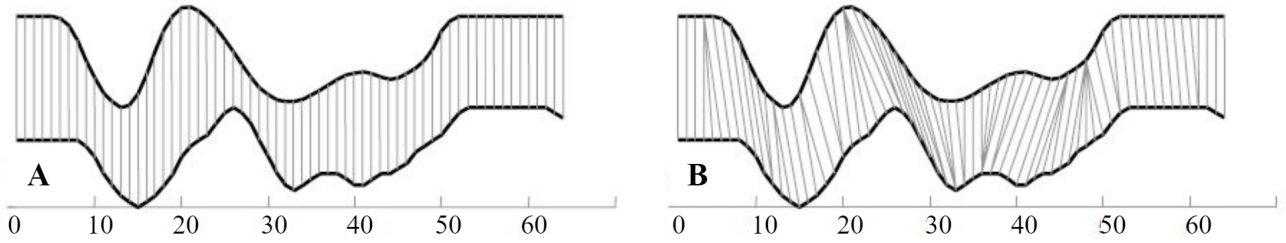


Figure 19: The DTW algorithm locally aligns the signals before calculating the (Euclidean) distance between them [4]

The often-cited drawback of the high time complexity of the DTW algorithm [166] has been largely eliminated thanks to algorithmic optimizations developed over the past decade, making it feasible for many practical applications [163, 167]. The computational complexity is no longer a bottleneck. Instead, the continued challenge of using the DTW-algorithm for real-world application lies in adjusting its hyperparameters to get the right degree of elasticity of the distance measure. Empirical demonstrations show, that optimal constraint parameters depend on the data set and are typically determined through an iterative process of trial and error [168, 169] and requires understanding of the overlapping effects of the various parameters.

A novel near-paramater-free distance measure called matrix profile distance (MPdist) was proposed in 2017, that is able to handle the vast majority of common data issues in real-world data [170]. It does this, by comparing the constituent patterns of both time series and considering them to be similar if they share many of the same patterns. The invariance of this comparison to the location of the patterns in the time series allows the distance measure to ignore dropouts and spikes, repeated patterns and phase shifts in the signal. As its name implies, the MPdist is based on the matrix profiles of the compared time series. Recall, that the matrix profile P_{AB} encodes the presence of every constituent pattern of A in time series B. Likewise, P_{BA} represents the presence of every pattern of B in A. The combined matrix profile P_{ABBA} repre-

sents the presence of every pattern of A in B and vice versa. Intuitively, P_{ABBA} contains all necessary information for a pattern-based comparison of two time series. The largest value in P_{ABBA} corresponds to the most dissimilar pattern within the two sequences (referred to as time series discord), whereas the smallest value corresponds to the most similar pattern (referred to as time series motif) [170]. Clearly, neither one of these extremes is particularly useful as a similarity measure. Taking the smallest value results in little discrimination between most time series. This would be akin to measuring the distance between sentences in the English language based on their most similar word. Since most sentences contain words like ”a” or ”the”, these would all be equidistant [170]. Likewise, taking the maximum value would result in a distance measure that is susceptible to a single dropout or spike. Instead, the MPdist is defined as the kth smallest value in P_{ABBA}. For a discussion of this parameter k, the reader is referred to the original publications [170]. Suffice to say, that empirical demonstration has shown for the effect to be negligible for most real-world data. For this reason, the MPdist can be considered a parameter-free distance metric.

Below is a summary of the relationship of the three distance measures:

- Euclidean distance: two time series are similar if the difference of every point along both time series is small.
- Dynamic Time Warping distance: two time series are similar, if after non-linear warping of the time axis, they can be made similar under Euclidean Distance [170].
- Matrix Profile distance: two time series are similar if their constituent subsequences are similar under Euclidean distance.

To be precise, both the DTW distance and the MPdist are distance *metrics*. The difference between a metric and a measure is that the latter does not conform to the triangle inequality i.e. if A is close to B and B close to C, A need not be close to C. All three of these distance measures are used by the anomaly detection algorithm presented in the next section.

4.2.3 Time series snippets

One of the most fundamental tasks of analyzing time series data is identifying typical, repeated patterns that are representative of a given data set. Finding such patterns is often extremely challenging for large data sets, primarily because of the difficulty of defining what constitutes a

representative pattern [147]. One apparently obvious definition is patterns that are frequently repeated in the data. A basic algorithm for identifying frequently occurring patterns (called motifs) in time series data was formalized as early as 2002 [153]. An (inefficient) motif discovery algorithm can be formulated based on the previously introduced concepts of time series joins and the matrix profile. Given a time series T, the algorithm applies a similarity join on the time series and itself J_{TT}. Subsequence pairs whose distance exceeds a given range R in the matrix profile P_{TT} are discarded. The remaining subsequences are ordered according to their frequency, with the most frequent subsequence called the first motif, the second most frequent subsequence called the second motif and so on. Note, that the motif discovery algorithm can be generalized to a *set* of time series objects instead of a single time series [157]. Essentially, the motif discovery algorithm performs a range similarity query for each subsequence in A and ranks these subsequences according to the number of returned results. While this brute-force approach is quadratic in time complexity, algorithmic optimizations have been developed to reduce the runtime by three orders of magnitude [5], making it feasible for real-world applications. Over the years, motif discovery has been applied in various domains like medicine, entertainment, biology [95] and manufacturing [5] (cf. figure 20).

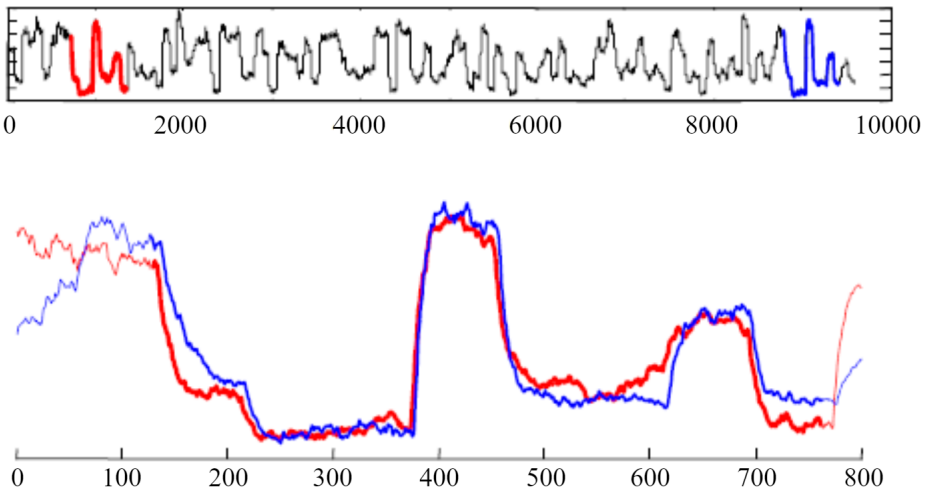


Figure 20: Motifs discovered in an industrial data set [5]

While motifs are useful for detecting and analyzing repeated structures in time series, they do not consider the *coverage* of the data. Coverage refers to the extent to which the data is represented by those patterns [171]. This makes them unsuitable for constructing an anomaly detection model. To detect anomalies, it must be possible to explain every part of a time series i.e. it should be possible to use the patterns stored in the model's library as building blocks to reconstruct the time series. In 2018 Keogh et al. introduced a data mining primitive

called time series snippets, that allows to extract patterns that guarantee complete coverage of the time series. Since snippets are fundamental to the high-level anomaly detection approach proposed in the next section, a brief overview of the algorithm is provided. It combine all of the previously introduced concepts of similarity joins, the matrix profile and the MPdist metric. The first step is to break the time series T down into a set of non-overlapping subsequences. These are referred to as snippet candidates S. The goal of the algorithm is to find an ordered set of snippets, that together cover the entire length of the time series as effectively as possible. The snippet finder calculates a distance profile between each snippet S and the time series using the MPdist metric. The resulting distance profile takes on low values along sections of the time series that share constituent patterns with the query and takes on high values otherwise. Consider the example of real-world tightening data in figure 21.

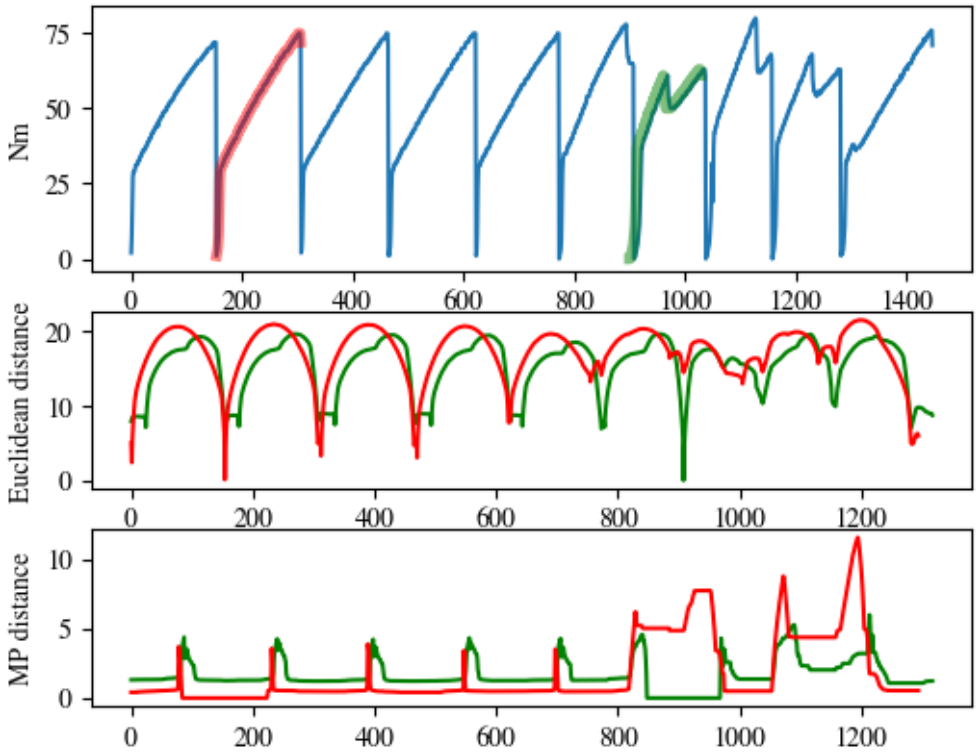


Figure 21: Distance profile using the MP distance and Euclidean distance for two randomly selected subsequences in manufacturing data.

When looking at the MPdist profile of a single snippet, the lower the area under the curve (AUC), the better its coverage of the data. This idea can be extended to a set of snippets by taking the *minimum* of their individual MPdist profiles at each time step.This curve is called the minimum (MPdist) distance profile. Based on this idea, it is possible to formulate a loss

function to find the set of snippets that best represents the data: minimize the AUC under the minimum distance profile [147]. Given a number of snippets k to select from p profiles yield k-choose-p possibilities. An exhaustive combinatorial search of all possible combinations of k snippets will quickly become intractable. To keep the runtime of the algorithm manageable, the snippet finder uses a greedy search strategy to find an approximate solution. The snippet algorithm will serve as the starting point of the unsupervised anomaly detection model proposed in section 4.3

4.2.4 Time series shapelets

The previous section discussed the unsupervised extraction of patterns that are representative of a large amount of unlabeled data. Additionally, it is often desirable to extract patterns that are maximally characteristic of a class of labeled data. These patterns can be used to classify an objects as belonging to a particular target class or discriminating an object as *not* belonging to that class. The shapelet algorithm published by Ye et al. in 2009 [146] was developed for this very purpose. The algorithm extracts maximally discriminative time series subsequences (shapelets) from annotated time series objects by evaluating the ability of each subsequence to separate the classes. Like the snippet algorithm, it is based entirely on the similarity join operation discussed in section 4.2.1 and therefore considered a data mining primitive (fundamental algorithm). Since its inception, it has been applied to numerous real-world settings for time series classification [75]. Due to its effectiveness, it continues to be subject of ongoing research interest and improvement efforts [172, 173, 174]. This section gives a very brief overview of the algorithm, as it is the basis of the semi-supervised template extraction and refinement process described in section 4.4. For a comprehensive description, readers referred to the original works of its authors [146].

Given a set of n time series objects T that belong to one of two classes, the shapelet algorithm evaluates each constituent subsequence to find the one that is maximally characteristic of one class and maximally uncharacteristic of the other class. This subsequence is referred to as a time series *shapelet* and can be used to classify an object as belonging to the same or a different class as the time series from which it was extracted. The first step is to extract a subsequence set A of subsequence length m for each of the n time series objects T. Initially, every subsequence is considered a potential shapelet *candidates* C. For each candidate C, the algorithm calculates the minimum distance between the candidate and every time series T i.e.

the matrix profile. The algorithm repeats this process for each candidate C and object T in the data set. The results can be used to quantify the ability of C to separate the classes. This can be seen by plotting the values on a number line as shown in figure 22.

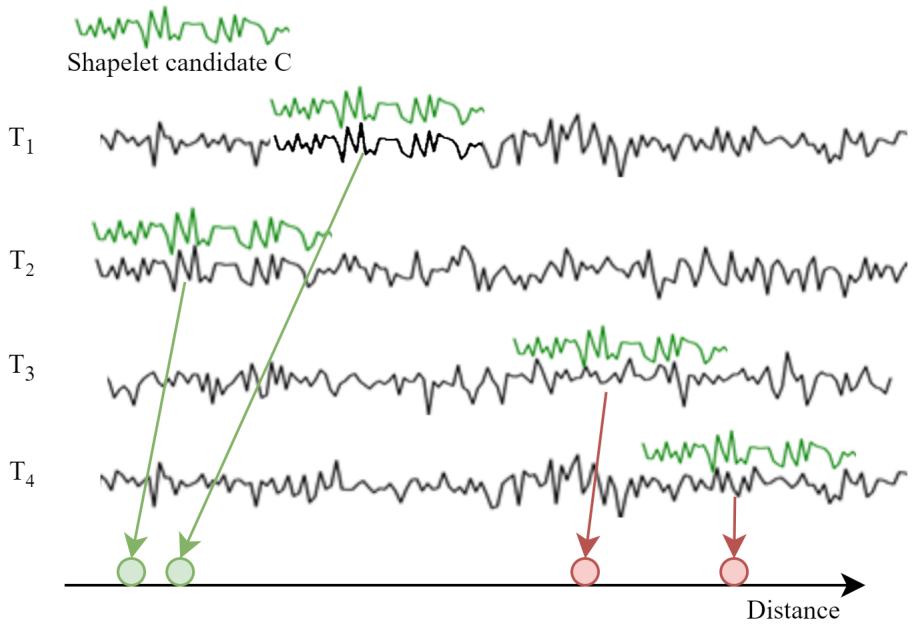


Figure 22: The class-separability of a shapelet candidate C can be evaluated based on the distance of that candidate to the nearest neighbor to every object T.

The algorithm scores the candidate based on the information gain criterion, which measures the ability to linearly separate both classes. The information gain criterion is based on the concept of information entropy, whereby a data set of high class purity has a low entropy and vice versa [175]. When linearly splitting a data set into two partitions, the total entropy either stays constant or decreases. The maximum reduction in entropy occurs for the split-point that separates the data set into partitions of maximum class purity. Thus, the criterion makes it possible to *quantify* the ability to linearly separate two classes. It is a concept from the field of information theory and widely implemented in many conventional ML algorithms to learn class-separating boundaries (e.g. the decision tree algorithm).

The distance calculation and evaluation is repeated for each candidate C, carrying forward the best-so-far shapelet candidate until all candidates have been evaluated. Intuitively, the final candidate with the highest score is a pattern that best separates the two classes. The shapelet algorithm essentially performs an exhaustive trial-and-error search of all possible patterns of length m in the data set. This brute-force approach requires calculating the matrix profile

between all subsequence sets A in the data set [149]. For a set of n time series objects of average length $\overline{m}$, the number of candidates of all possible lengths is $O(\overline{m}^2 n)$. Scoring each candidate takes $O(\overline{m}n)$ resulting in a total complexity of $O(\overline{m}^3 n^2)$ for the algorithm [176]. This is intractable for most real-world data sets [176]. To reduce the number of calculations, algorithmic optimizations have been developed such as (1) caching the distance calculations during the similarity join operation [177], (2) abandoning the distance calculation between a candidate and a subsequence if it exceeds its lowest-so-far distance to the time series (early abandoning) and (3) abandoning the distance calculation as soon as the maximum possible information gain falls below that of the best-so-far shapelet (entropy pruning). Their combined effects is a reduction in the time complexity by approximately three orders of magnitude, resulting in a runtime of a few hours for thousands of time series [176, 177]. Additionally, simple dimensionality reduction methods like piece-wise aggregate approximation (PAA) allow additional speed-ups in the order of a magnitude [145] (essentially, this reduces the average time series length $\overline{m}$). These algorithmic optimizations are sufficient to make the algorithm feasible for many real-world applications [8].

4.3 Unsupervised anomaly detection

This section presents an approach that uses the snippet algorithms discussed in section 4.2.3 to construct an unsupervised anomaly detection model. The first subsection explains how the algorithm is adopted from its standard implementation. The second subsection discusses how the snippets are organized and used for anomaly detection. The focus of this approach is to ensure a highly orchestrated training process and easy debugging of the model by data scientists. As discussed in section 2.1 this is vital for scaling and maintaining a large-scale system. The process of adapting the pattern library based on user feedback and transforming model into a true one-class model for anomaly detection is explained in section 4.4 and section 4.5, respectively.

4.3.1 Adopting the snippet algorithm

The snippet algorithm can be used to extract representative patterns from unlabeled process data. Before discussing how a shapelet-based model for anomaly detection can be constructed, it helps to consider some practical aspects related to the application of the snippet algorithm. First, that the extracted snippets are, in fact, representative of normal process operation and do not capture any anomalous patterns. There are two ways to ensure that this is the case:

- SMEs can manually annotate a (small) subset of the available process data that is used for snippet extraction. This requires in a trade-off between a small subset to minimize SME resources and a subset large enough to be representative of the normal class.
- The number of anomalies in the data are generally very low. This is one of the main challenges that make it difficult to train ML models. In this case, however, this fact can be exploited by using a large subset of raw data in which the (expected) fraction of anomalies will be sufficiently low.

The second option is much more preferable as it allows to automate the training process without requiring manual input from SMEs. In practice, the size of the subset is selected so that it is *considerably* larger than required. This can be considered a form of weak supervision, as this training process is based on assumptions about the data (that is, that most data belongs to one class).

Second, the original snippet algorithm extracts snippets from a single time series. However, process data is typically recorded as separate time series objects for individual parts, batches, machine cycles, etc. Thus, snippets need to be extracted from a *set* of time series. At first glance, this may seem like a trivial problem of simply concatenating the time series and running the algorithm on this long time series as before. However, the concatenated time series will contain periodic pattern where the start and endpoints of consecutive time series have been joined together. This is depicted in red in figure 23. This pattern recurs often and will therefore tend to end up in the top snippets. Thus, the subroutine of the algorithm needs to be adjusted to extract snippet candidates C_S in such a way, that these patterns are avoided. This is done by placing a marker between each concatenated object and discarding snippet candidates C_S that contain this marker. While this avoids the extraction of undesirable shapelet candidates S_C it will lead to a low snippet coverage around these markers. To resolve this, the otherwise fixed snippet length m is dynamically adapted until the snippet candidates C_S fill the interval between two markers. This is schematically shown for real-world manufacturing data in figure 23.

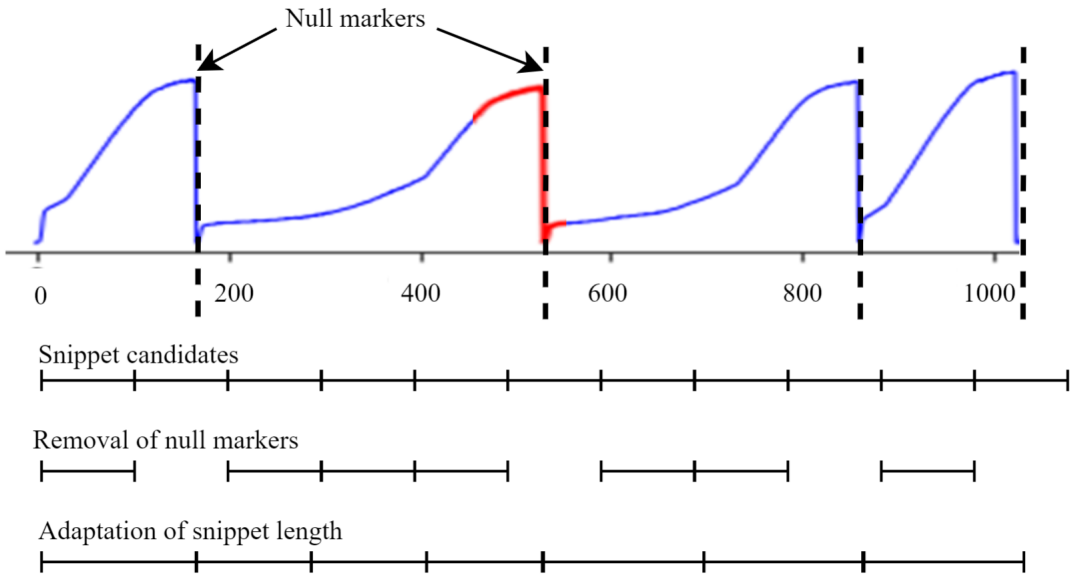


Figure 23: The snippet length m is dynamically adapted to optimally cover the time series between adjacent null markers.

Third, it is not immediately clear how to automatically set the hyperparameters of the snippet algorithm i.e. the (average) length and number of snippets. As discussed in section 2, a high degree of automation is desirable to ensure system scalability. Since snippet discovery should occur completely unsupervised, it must inevitably rely on a heuristic. The ability to quantify the snippet *coverage* makes it possible to define such a heuristic for the number of snippets. For a given snippet length, the (decreasing) coverage of the data for every additional snippet can be measured. By using a simple knee-point detection algorithm, an optimal number of snippets can be determined. Essentially, the knee-point detection is used as a cut-off criteria for the decreasing marginal benefit of adding an additional snippet. Knee-point detection is a widely adopted algorithm in machine learning typically used for determining an optimal threshold setting based on the ROC curve. Finding a good snippet size, on the other hand, requires a trial-and-error grid search for different lengths. Note, that the process of unsupervised model initialization need not be optimal. Since the aim is to adapt the model based on user feedback anyways, the unsupervised model must only provide a reasonable first that is good enough to start a feasible feedback cycle and overcome the cold-start problem discussed in section 2.

4.3.2 Anomaly detection using snippets

The previous section discussed the application of the snippet algorithm for the extraction of representative patterns from unlabeled process data. To understand *how* anomalies can be

detected using these patterns, it is necessary to define *what* constitutes an anomaly in the context of temporal pattern detection. There are three ways in which a time series can be considered anomalous:

1. The time series contains an anomalous pattern that *is not* representative of the normal class.
2. The time series is missing a pattern that *is* representative of the normal class.
3. The time series contains exactly the same patterns as the normal class but their position along the time axis varies.

This differentiation is a result of the need to set a fixed length for the snippet algorithm. All anomalies can be considered as belonging to category 1 if the anomalous pattern can take on any arbitrary length. To detect both the *presence* of anomalous patterns and the *absence* of normal patterns requires a comprehensive set of normal patterns that covers all eventualities. Essentially, the idea is that it must be possible to reconstruct a normal time series from this library of snippets. This does not mean, that every snippet is present in every object but that every (normal) object can be reconstructed form a *subset* of possible snippets. Consequently, a snippet-based model for anomaly detection cannot simply match all snippets to an object and rely on global thresholds for classification. Instead, the model must use thresholds for combinations of snippets. In principle, ML algorithms could be used to learn a multivariate decision boundary for all snippets. However, the resulting model will quickly become complex and difficult to interpret, especially for large snippet libraries. Instead, a hierarchical process of snippet matching is proposed that consolidates the results into a small number of meaningful features. This makes the final ML model simpler and easier to interpret.

Detecting anomalous patterns The starting point of the algorithm is a library of snippets extracted from a set of normal data using the approach described in the previous section. Given a new time series T, the first step of the algorithm is to calculate a distance profile D for each snippet S in the library using the MPdist. The algorithm then calculates the minimum across all of these distance profiles. This yields a meta-profile of equal length, that indicates how well each corresponding section of the target time series can be represented by *any* normal pattern. This meta-profile is called the minimum distance profile MDP. Note, that these steps are analogous to the snippet algorithm. Low values in the MDP indicate, that the corresponding

section of the target time series is similar to a normal pattern. Conversely, the maximum value of the MDP quantifies the maximum *dissimilarity* of the target time series to *any* normal pattern. Thus, the value $max(MDP)$ can be used to quantify the *most* anomalous pattern in the time series. This makes it possible to detect anomalous patterns in time series using a single threshold. This threshold can be learned by standard ML algorithm such as decision tree algorithm, SVM algorithm etc. This is schematically depicted in figure 24.

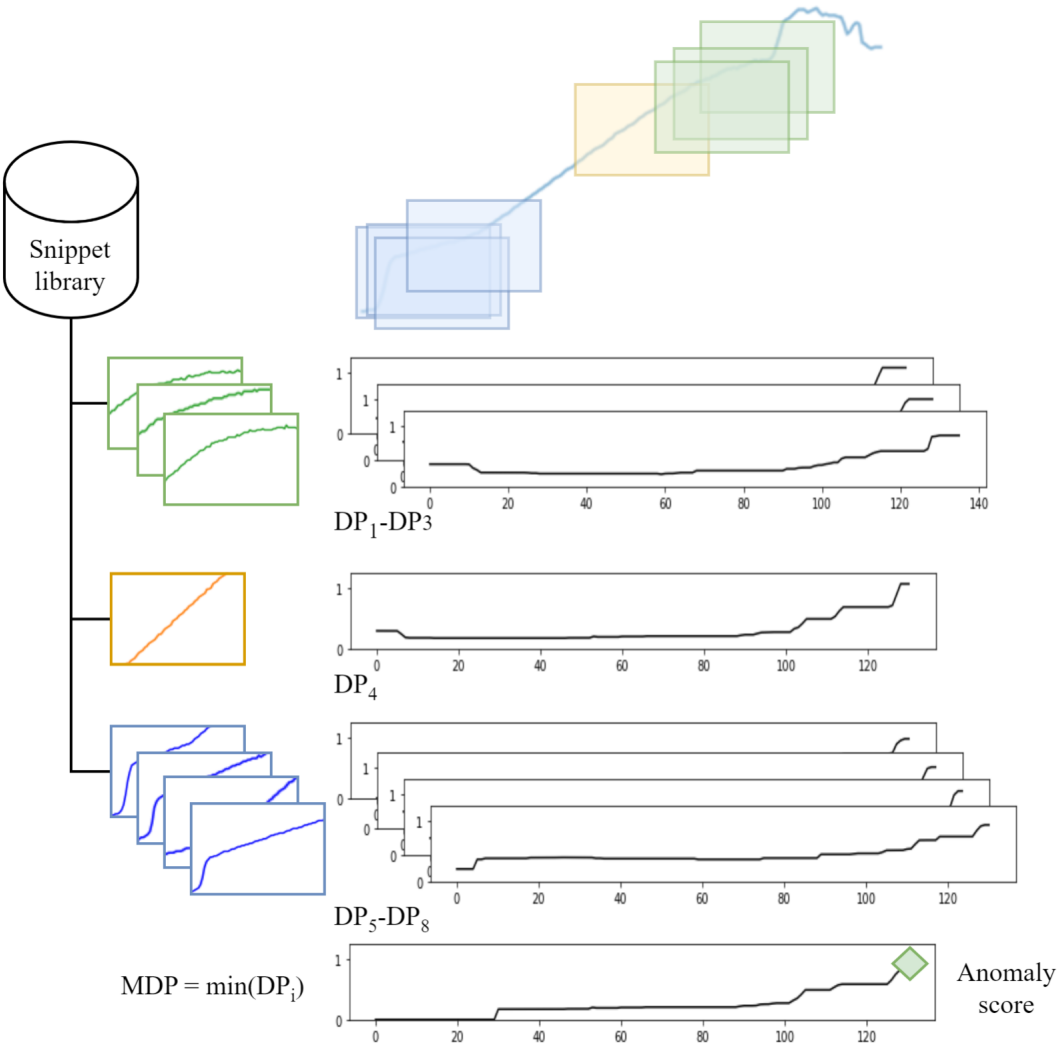


Figure 24: The value $max(MDP)$ quantifies the most anomalous pattern in the time series that cannot be explained using the current snippet library.

Detecting the absence of expected patterns The next step is to use the same library of snippets S to detect the *absence* of normal patterns. Once again, the starting point is the distance profiles DP between each snippet S_i in S and a time series T. The minimum value of the distance profile $min(DP_i)$ of snippet S_i quantifies the *presence* of the snippet in the time series, whereby $min(DP) = 0$ corresponds to a perfect match and a large value indicates the

absence of the snippet. Thus, the information about the presence/absence of a snippet S_i is captured by its value $min(DP_i)$.

The information contained in the values $min(DP_i)$ is consolidated into fewer, more meaningful features. To make these values comparable for different snippets, they must each be standardized. This is because the variance of the values $min(DP_i)$ may differ for different snippets. This metadata is stored in the snippet model. If all time series objects were to contain *all* patterns in the library S, the maximum value of all distance profiles $max(min(DP))$ could be used to quantify the absence of *any* normal pattern. However, this is generally not the case for real-world data. Manufacturing processes often exhibit various normal operating modes, characterized by distinct, often mutually exclusive patterns. This may be due to a host of secondary influences such as variable machine settings, material properties, etc. Many of these variations will be captured in the snippet library but are only present in a (significant) minority of the objects. Thus, time series of the normal class need only contain some patterns to be considered normal. Figure 25 shows a library of snippets extracted from a tightening process that exhibits two different operating modes (corresponding to different machines). For a particular time series object, either snippet A or B will be absent, although both are representative of the process.

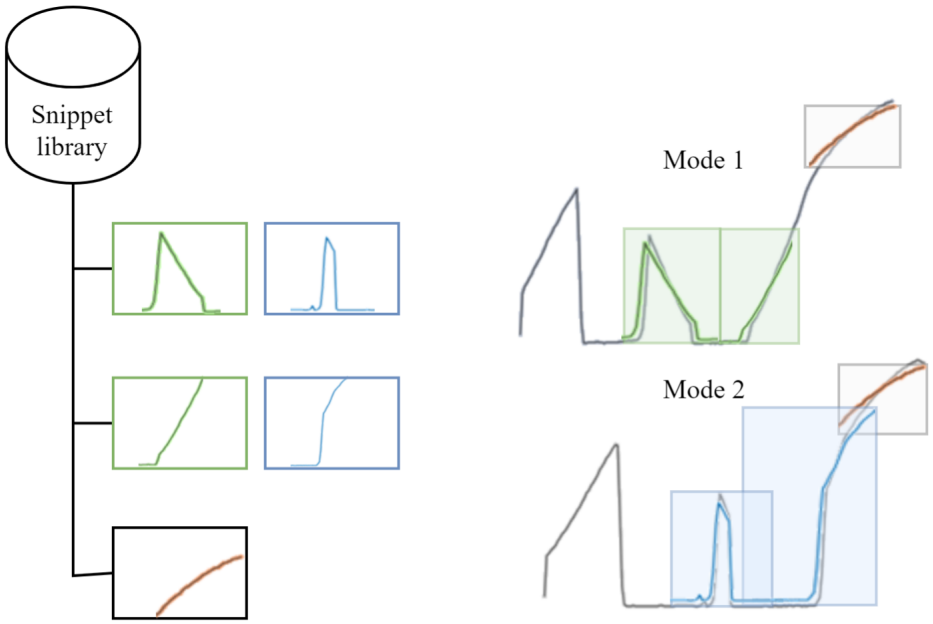


Figure 25: An excerpt of a snippet library for a tightening process that are representative of mutually exclusive operating modes.

Therefore, the snippet library must be subdivided into subsets of mutually exclusive patterns so that a particular time series object contains exactly one pattern from each subset. These subsets are defined based on the same training data from which the snippets were extracted. Analogous to the snippet algorithm, the objects are divided into non-overlapping sections. Then, each snippet S_i in the library is matched to each time series T and the matching section recorded. If two snippets are matched in the same section across different time series, they are assigned to the same subset. This meta-data is stored by the model and used to determine the minimum value for each *subset* of snippets $min(min(DP))$. This values indicates if any one of a number of alternative patterns is present for a subset. Figure 26 shows how this metadata is used to consolidate the features.

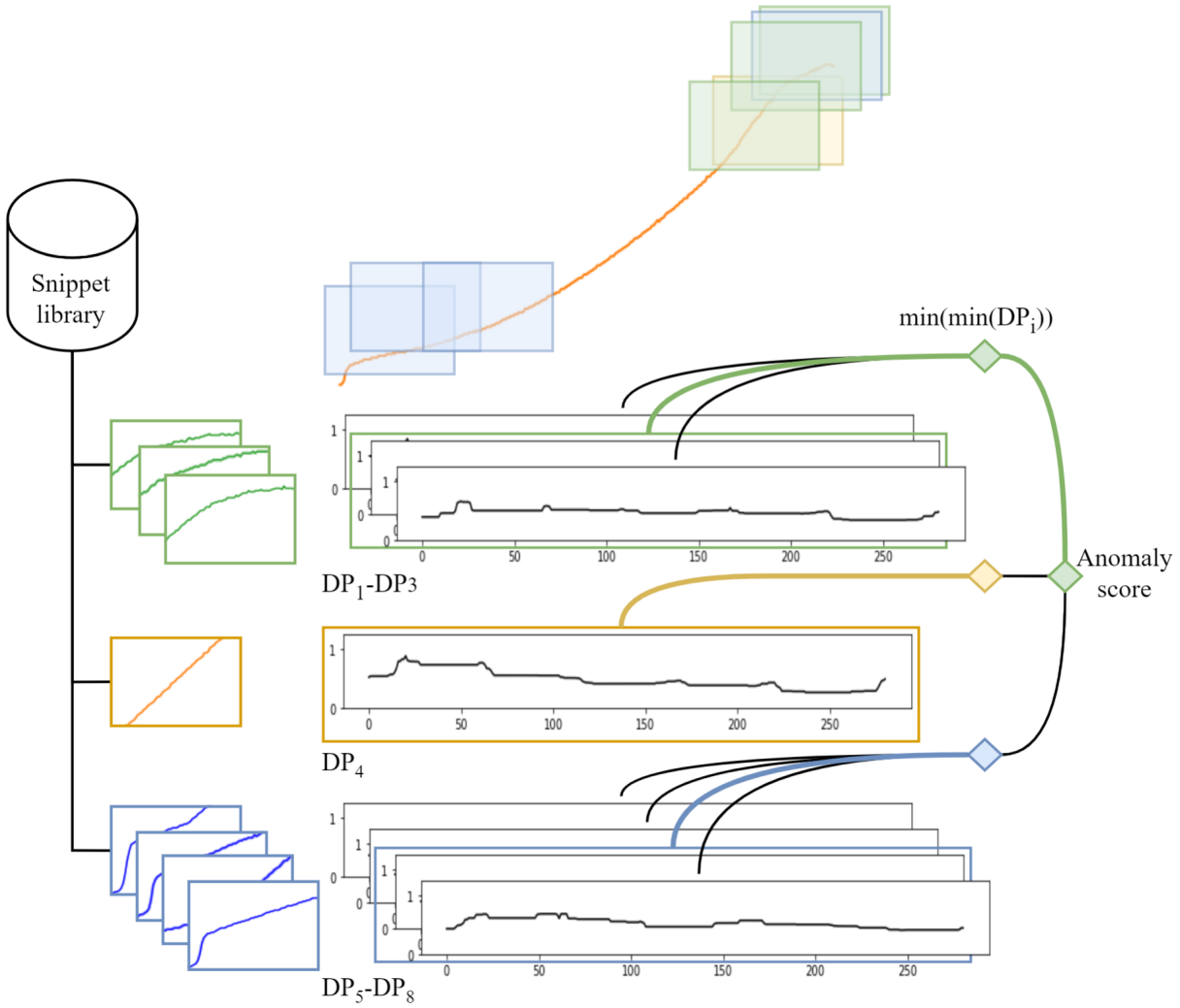


Figure 26: The patterns in the snippet library are matched to the target object and the results successively consolidated across snippet subsets.

Thus, the ML model must only learn simple thresholds i.e. univariate decision boundaries for each subset. This results in a vastly simpler model. Additionally, the individual scores can be further simplified into a single high-level score $max(min(min(DP_i)))$. This *absence score* indicates the absence of a normal pattern in any of the subsets. Together with the novelty score presented in the previous section, it can be used by data scientists as a high-level approach to visually inspecting the ”feature space” of the model.

Retracing model decisions The snippet-based model discussed so far makes it possible to quantify both (1) the presence of anomalous patterns and (2) the absence of normal patterns in a target time series (at the expected point along the time axis) using only two anomaly-scores. These scores are based on a two-step hierarchical decision process, that uses simple logical operations to evaluate the ability to reconstruct a time series from a library of normal patterns

(snippets). By using a threshold, these scores can be directly used for anomaly detection. To learn these thresholds directly from the data, existing unsupervised ML algorithms can be used. The result is an inherently interpretable anomaly detection model, that facilitates communication between data scientists and SMEs and more targeted model debugging and improvement. To understand what this looks like in practice, consider the same anomalous objects discussed in the beginning of the section. Figure 27 shows how these scores allow to detect the previously discussed anomalous objects. When the two anomaly scores are plotted on a plane, we can see that they complement each other well.

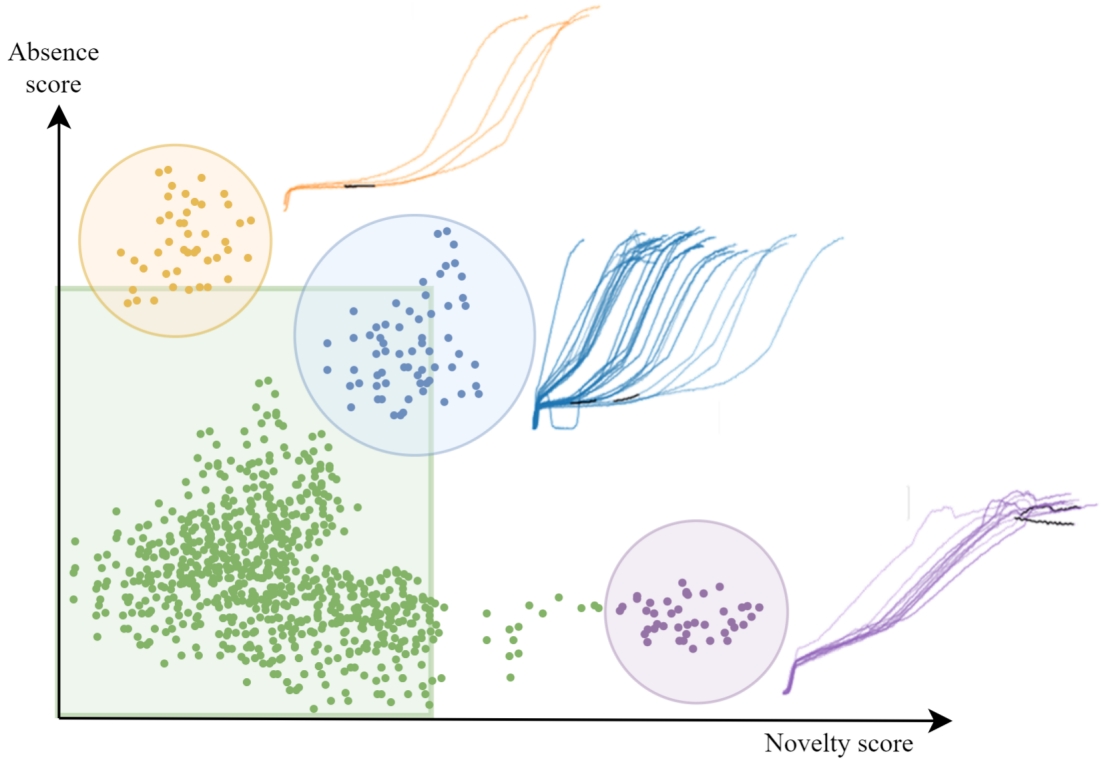


Figure 27: The high-level algorithm based on time series snippets allows intuitive clustering.

The great advantages of the proposed approach is the interpretability of the model that makes it easy to debug. This enables data scientists to investigate the cause of poor model performance. Consider object A in figure 27. This object is classified as anomalous by the model but does not belong to a target class. To to understand why, data scientists can simply retrace the decision process of the model as schematically depicted in figure 28. While the normal class exhibits a characteristic bend between the transition of the two linear compression phases, object A does not. Instead, it exhibits a smooth, progressive increase in the tightening torque. Since this phenomenon is rare, the snippet algorithm did not extract and match a corresponding snippet.

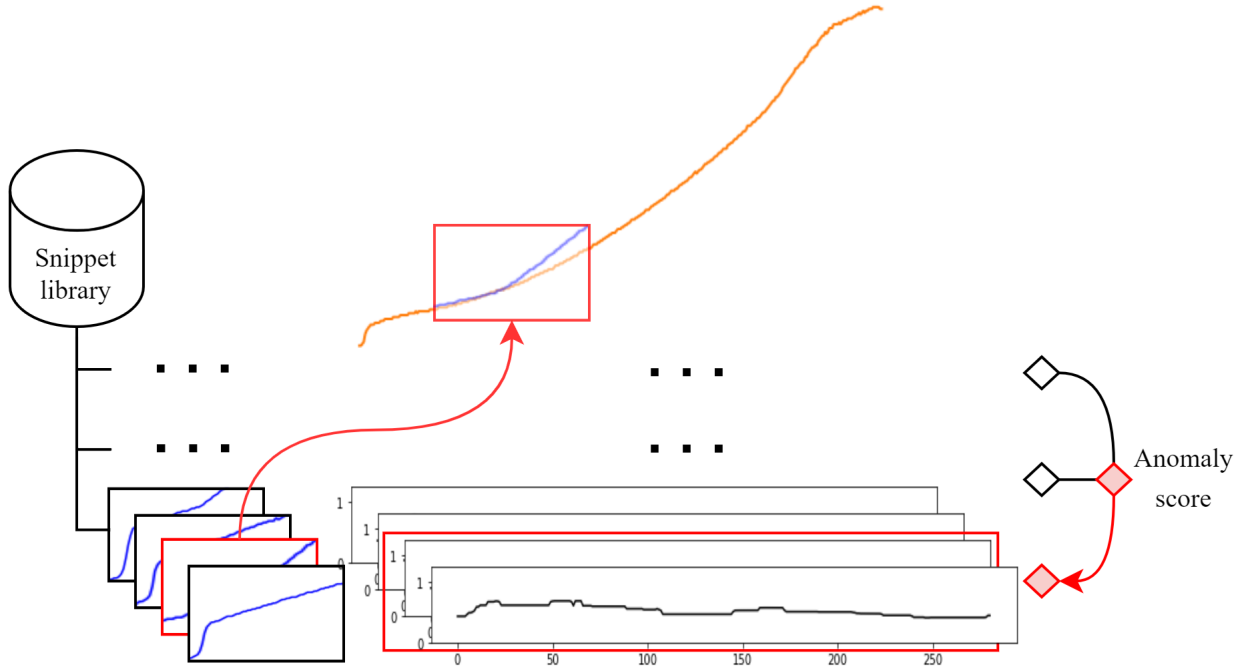


Figure 28: Data scientists can retrace the decision process of the snippet model to understand why a particular object was misclassified

4.4 Semi-supervised pattern extraction and refinement

The previous section introduced an algorithm based on time series snippets for the unsupervised detection of anomalies. These anomalies are then manually annotated by SMEs. The next step is the extraction of representative pattern from this annotated data. These patterns are used either for (1) classifying a particular target class or (2) updating the one-class model to ignore uninteresting anomalies in the future. The latter is discussed in detail in the next section. The first question that needs to be addressed is how to extract these representative patterns. The amount of annotated data is typically very small because anomalies are inherently rare and the resources of SMEs for annotating the data severely constrained. This section proposes ways to address these challenges. The first subsection presents an improvement of the shapelet algorithm discussed in section 4.2.4. This adaptation makes it possible to extract robust shapelets from highly imbalanced time series data [6]. The second subsection proposes an information retrieval system that enables SMEs to efficiently sieve through large time series data archives to annotate historical data of a target class. Throughout the field-testing and development stages of this research, this guided annotation of the data archives has proven to be an extremely cost-effective way of improving the classification accuracy of the deployed ML models (and simultaneously improving SME understanding of the data) [167]. Both approaches

are semi-supervised as they make structural assumption about the underlying data. Due to their broader relevance outside the manufacturing domain, both approaches were validated using benchmark datasets and independently published during the course of this research [8, 6]. This section includes some results of these benchmark studies in addition to field testing results from the manufacturing domain.

4.4.1 Extracting shapelets for highly imbalanced data

This section summarizes changes to the shapelet algorithm introduced in section 4.2.4 that improve the algorithm's applicability to highly imbalanced data. These change allow to extract more robust patterns using fewer annotated objects at the expense of a considerable increase in runtime. Nonetheless, this trade-off is acceptable for many real-world applications.

The shapelet algorithm described in section 4.2.4 yields a single shapelet. Empirical studies have shown, that the accuracy of shapelet-based classifiers can be significantly improved by using multiple shapelets [178]. This is unsurprising, as drawing on a larger number of features allows a model to make more complex decisions (or be more certain of a decision). Moreover, some classification problems *require* the consideration of multiple features simultaneously [7]. However, shapelet sets frequently contain a large number of redundant and non-robust shapelets. This has adverse effects on the accuracy and interpretability of the classifier [178] and is attributable to two causes:

- The shapelet selection process is based on a univariate evaluation of each shapelet candidate. However, taking into accouunt the redundant information shared by multiple features is often necessary to select non-redundant shapelets.
- Existing methods for evaluating a shapelet candidate are based on its *ability* to linearly separate the classes without considering the *margin* of the class separation. The margin is widely recognized as an important indicator of feature robustness [179] and is particularly critical for imbalanced data [176].

As long as enough labeled data is available, these drawbacks can be largely compensated by extracting a large number of shapelets and eliminating feature redundancies through appropriate adjustments to the training process like filtering or cross-fold validation. However, this approach is not feasible in the case of highly *imbalanced* data, where only a handful of instances of the target class are available.

The challenges of extracting a shapelet set This section gives an overview of the challenges of extracting a set of shapelets from (imbalanced) data. It starts with a brief introductory discussion of the shortcomings of existing approaches before detailing proposed changes that improve its ability to find robust shapelets with minimal information redundancy. The shapelet algorithm scores every candidate, keeping the best-so-far candidate. Thus, it seems as though finding a set of shapelets is straightforward: simply keep the top-k candidates instead of the single best candidate. However, a shapelet set constructed this way would contain a large number of overlapping shapelets of the same feature. The information redundancy would render the shapelet set useless, even for moderately sized data sets. One approach to resolve this issue is to split the data using the optimal split-point of the final shapelet and reiterate the shapelet discovery algorithm on the subsets. This approach is proposed by the original authors for multi-class classification problems [146] and can be adapted as a recursive splitting process for binary classification problems. However, this approach of essentially constructing a shapelet-based decision tree will return suboptimal shapelet set for two reasons:

- Approximating a multivariate decision boundary via univariate split-points is inefficient [6]. This is an inherent problem of univariate feature selection and results in a set of redundant shapelets and consequently inefficient models that are difficult to interpret [178]

- A recursive shapelet search is only possible as long as the subsets contain objects of both classes i.e. no ”perfect” classification is possible based on the current shapelet set. Once all objects of the target class can be linearly separated, the algorithm terminates. While this may seem logical, there are often additional candidates that could be included in the shapelet set to improve robustness. This is problem is much more pronounced for highly imbalanced data sets where it is often possible to ”perfectly” separate a few objects from the majority by focusing on a spurious distinguishing features that does not generalize to new data.

Recognizing the advantages of a shapelet set, Hills et al. proposed to transforming time series data directly into a shape-based feature vector, using a set of k shapelets [178]. The main objective of this work was to decouple the shapelet extraction from the decision tree classifier and establish it as a stand-alone method of feature extraction. This made it possible to combine the advantages of shapelets with the advantages of more powerful ML classification algorithms. To address the previously discussed problem of information redundancy in the top-k shapelets

returned by the original shapelet algorithm, the authors proposed a pruning strategy to exclude overlapping shapelet candidates from the same time series from the shapelet set. While this reduces the information redundancy in the shapelet set, it is unlikely to yield an *optimal* set, as it only considers the feature redundancy within an individual time series. Since multiple time series in a data set often contain the same features [180], redundant shapelets may still be added to the set from different time series. This effect has been documented in the original paper [178] and becomes more pronounced as the imbalance and uniformity of the data increases [6]. To address this problem, Grabocka et al. extended the pruning strategy to consider the similarity between the next-best candidate and the best-so-far candidates in the shapelet set [180]. However, neither of these approaches considers the multivariate dependencies between the shapelet set and the next-to-add candidate [6].

Margin-based shapelet scoring The shapelet algorithm scores each shapelet candidate based on the information gain criterion. This entropy-based split-point criterion is simple to compute well-suited for scoring shapelet candidates for most applications. However, its inability to consider the separating *margin* may sometimes lead to undersirable results. Consider the two minimum distance vectors of candidates A and B in figure 29 whose optimal split-points lead to the same class purity in the subsets. The information gain criterion would yield the same score for both candidates. However, candidate B intuitively looks like a more robust choice that we would expect to generalize better to unseen data. This is because it exhibits a lower inter-class variance and a larger intra-class separating margin [6]. The margin is an important indicator of feature robustness [181, 179] and is the fundamental principle underlying margin-based ML algorithms. It is often used to estimate the generalization error of the classifier [182] and its ability to generalize well to unseen data [179]. Although this information is available in the data, the information gain criterion makes no use of it. Consequently, the shapelet algorithm indiscriminately selects either feature.

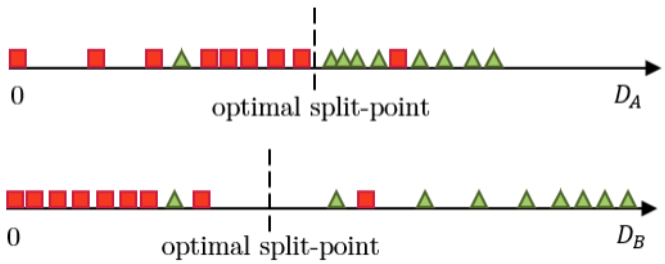


Figure 29: The values of the minimum distance profiles D_A and D_B of two shapelet candidates A and B plotted on separate number lines. The maximum information gain for both optimal split points is the same [6].

This phenomenon, while generally undesirable, is particularly problematic for imbalanced data. Measures of class purity are ill-suited for evaluating the separability of a highly imbalanced minority class, as it is biased towards the majority class [183, 184]. This is why decision tree algorithms are prone to overfitting when trained on imbalanced data, resulting in deep and unstable trees [184]. To counteract this problem, cost-sensitive learning strategies are often employed, where the cost of misclassification is adjusted to reflect the class imbalance (classification errors of the minority class are penalized more strongly to counteract the bias). For the shapelet algorithm, replacing the information gain criterion with the F-score (which emphasizes the misclassification error) improves the accuracy for many data sets [178]. However, neither score makes use of the (multivariate) margin of the class distributions within the shapelet feature space. This is the main caveat of any entropy-based scoring method, as they ignore the *margin* of separation i.e. how far these distributions (specifically the tails of these distributions) are apart.

Based on these considerations, a new way of scoring shapelet candidates is proposed that replaces the entropy-based split-point with a margin-based split-point. The idea of utilizing the margin concept for shapelet scoring is not, in fact, a new one. It was mentioned in the original shapelet discovery algorithm as a tie-breaker for shapelet candidates that have the same information gain [176], whereby the margin is approximated as the difference between the class means (a crude, computationally cheap way to approximate the margin). Essentially, the idea is to reverse the order of operation, by using a margin criteria to *find* the most suitable split-point and the information gain criteria to *evaluate* the shapelet candidate using the margin-based split-point. The advantages are, that (1) the split-point considers the margin of separation, improving the applicability for highly imbalanced data and (2) the margin can be extended to the multivariate case, ensuring that complementary shapelets with low information

redundancy are selected in a form of ”simultaneous feature extraction and selection” [6]. This makes it possible to quantify and automate the feature selection process. To demonstrate this effect, we can compare the margin-based scoring to the conventional scoring approach on the public Gun Point benchmark data set. This is one of the most widely studied data sets in literature [176] and published results are available for both the shapelet discovery algorithm and the shapelet transformation algorithm [146, 178]. This makes it easier to validate the results of our work. Fig. 30 depicts the feature spaces of the top shapelet selected by both scoring methods for the test data.

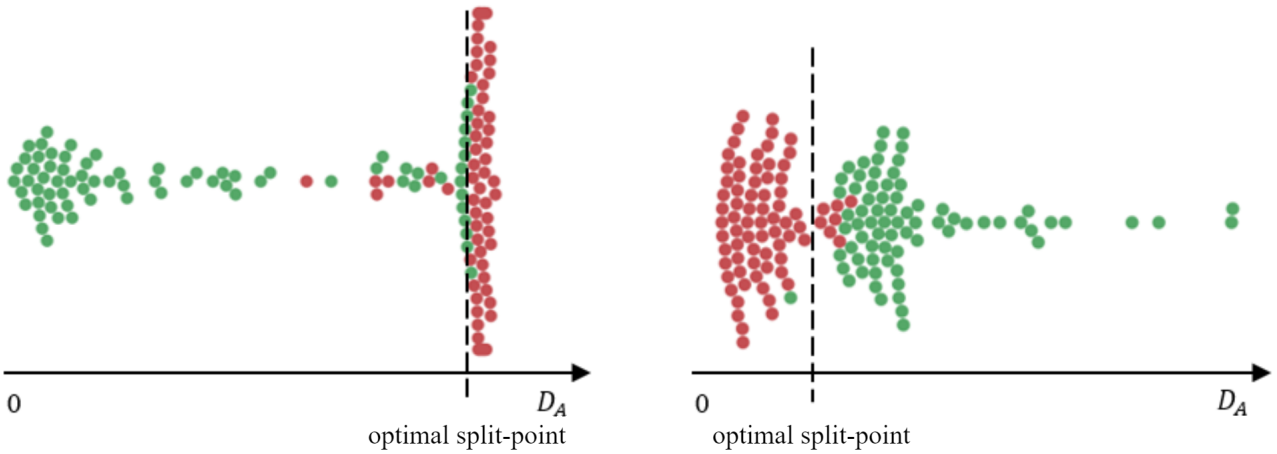


Figure 30: Swarm plot of the feature space of the top shapelet for the conventional maximum information-gain criterion (left) and the proposed margin-based criterion (right) [6].

The two scoring functions select shapelets with a vastly different feature space (in particular, thetop-k shapelets are extracted from opposite classes). While the margin-based shapelet does not allow to linearly separate the classes on the training set, this is (just) possible using the shapelet found via the conventional information gain criterion. This is a consequence of the (soft) margin implemented in the algorithm, that may lead to imperfect accuracy even if the train set is linearly separable [6]. While on the face of it, this may seem like a disadvantage, it is in fact a *desirable* property. For many (particularly imbalanced) classification problems, enforcing a hard decision boundary can lead to overfitting of the data and low generalizability to new data. This is evidence by the margin-based split-point that performs more robustly on the test set than the conventional split-point criteria, achieving a much higher accuracy. Additionally, the sensitivity of the classification accuracy to changes in the split point is much lower. As discussed, this improved generalizability of margin-based decision boundaries is particularly important in the case of imbalanced data.

Greedy shapelet selection In addition to improving the robustness of the shapelet, the margin-based shapelet scoring is fundamental to extracting a shapelet *set* that (1) consists shapelets with minimal information redundancy to improve classification accuracy, (2) is small to improve the interpretability of the classifier [178], and (3) whose quality can be quantified as more shapelets are added to the set. As discussed in section 4.4.1, any *univariate* evaluation of shapelet candidates will tend to select redundant shapelets to approximate a multivariate decision boundary. To avoid this, the class separability of a combination of feature must be evaluated *simultaneously*. This requires a multivariate decision boundary in the feature space. The margin-based decision boundary can be simply extended to the multivariate case to evaluate the entire shapelet set. For n shapelet candidates, the number of possible combinations of size k (permuatations without replacement) is determined by the binomial coefficient $\binom{N}{k} = \frac{n!}{k!(n-k)!}$. The brute-force method of finding an optimal solution to this problem is intractable for the large number of shapelet candidates n. The problem, therefore, is which *combination* of shapelet candidates to consider. To resolve this issue, a greedy search strategy is used, that builds the shapelet set step-by-step. This is a common approach to finding an approximate solution to otherwise unfeasible combinatorial problem. Each candidate is evaluated in combination with the current shapelet set and the best one added to the set. This is repeated one-by-one until k shapelets have been extracted. Fig. 31 shows the top five shapelets returned by the margin-based greedy shapelet search [6] and the widely cited shapelet transformation [178]. While in the latter case, all shapelets are concentrated in similar sections of different time series objects (representing the same/similar feature2), the shapelets greedy search returned a shapelet set that (qualitatively) seem to capture *additional* features. A benchmark study conducted for a number of popular ML algorithms supports this observation, enabling equal or higher classification accuracy for all models.

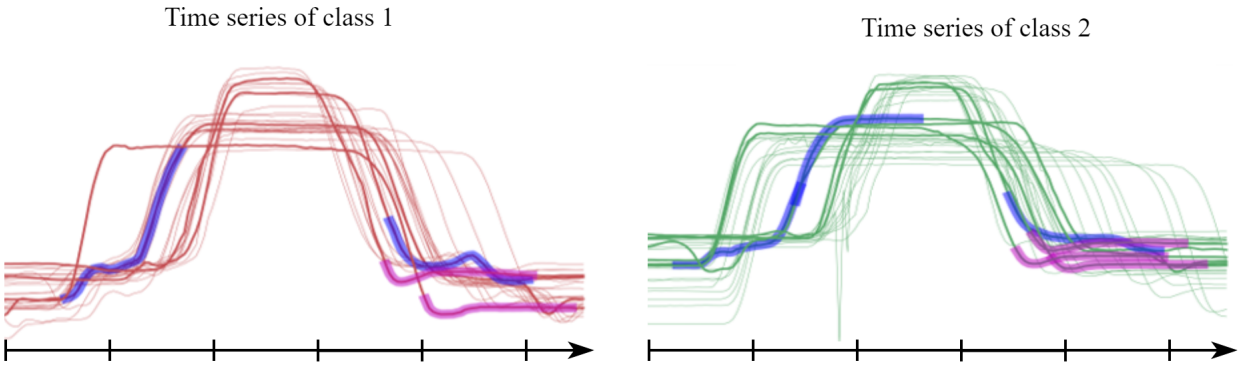


Figure 31: Top five shapelets extracted using the shapelet transformation algorithm (magenta) vs. the margin-based greedy shapelet search (blue) [7].

Additionally, a termination criterion for the greedy search that is based on the growing combined margin of the shapelet set is proposed. As shapelets are added to the set, the multivariate margin will increase asymptotically (albeit not necessarily monotonically, since a greedy search only finds the locally optimal solution at each step). When the incremental margin Δm growth is small, the search can be terminated since adding more shapelets will not have a significant impact on the classifier. The main advantage of this criterion is that it replaces the use-defined parameter k with the parameter Δm. While this may seem like trading one (bad) thing for another, it is a much more objective stop criterion gives an indication of (1) how well the classes can be separated with the shapelets extracted so far and (2) how much more information there likely is in the data. While there is no universally applicable optimal value of Δm, it is much more steady across data sets/within a setting than k. During field-testing in the manufacturing setting described in the section 6, the empirical results showed that most classifiers achieved acceptable accuracy when the search was terminated once Δm fell below 10 percent. While this value is not universal (due at least in part due to the fact that there is no universally acceptable limit of what constitutes ”acceptable” model accuracy), it suggests that Δm can, at the very least, be used as a data-specific heuristic to determine the size of the shapelet set. This is important when trying to automate model training. Additionally, the Δm-criterion tends to produce very small feature sets. This is desirable, because (1) the complexity of the greedy search (unlike existing algorithms) scales linearly with k - keeping this number small avoids unnecessary resources and (2) fewer features generally result in simpler models that are more interpretable [178]).

4.4.2 Interactive data exploration and pattern refinement

During the relevance cycle of the anomaly detection algorithms, it became clear that (1) more annotated data (in addition to the detected anomalies) could decrease the time spent on model tuning considerably and (2) historical data stored in data archives could be a readily availble source of this data. This insight initiated a secondary design cycle for the development of a search system to retrieve similar objects from large time series databases. This novel information retrieval (IR) system constitutes a stand-alone research artifact useful for many related applications (cf. section 1.3.2) and was therefore published in a domain-agnostic journal [8]. During the subsequent relevance cycles, it has emerged as an important way for increasing system interaction and data understanding of SMEs and reducing the workload for data scientists.

In many large-scale manufacturing settings process data is recorded for each part during production and archived in large time series databases [2]. In practice, there are a number of scenarios where these archives can serve as an important source of information:

- If an interesting anomaly is detected, a retrospective search through the archives may reveal further affected parts. Accurately identifying these allows OEMs to contain the damage and launch a targeted recall [8].
- Searching for objects similar to annotated data allows SMEs and data scientists to increase the amount of annotated data and improve model performance.
- SMEs may wish to better understand an anomaly. Often, the casual mechanisms behind an anomaly are initially unknown. Finding similar instances in the past allows SMEs to cross-reference change in the environment and occurence of the anomaly and establish a lead into potential causes.

When searching through the database, the user wishes to retrieve objects that are similar to a given search object of interest. This is known as similarity search. Often, the object of interest is an anomaly detected by a model in production. The main challenge that complicates this search is the size of the data archive. For a manufacturing database, the search quickly encompasses millions of time series objects, even for short time frames of only a few days [8]. An exhaustive visual inspection of the data is therefore intractable. Instead, automated information retrieval systems are required to guide and conduct this search (similar to the way search engines are

[2] For critical processes, OEMs may be required to do so by law.

used to search the Internet). This section presents a novel information retrieval system to efficiently search through large time series databases. It combines a relevance feedback cycle with automated query adaptation that makes use of the optimized shapelet algorithm presented in the previous section.

Exploratory search in databases Typically, the interaction between a user and the database in question is exploratory in nature, owed to the imperfect understanding of both the data (the search space) as well as the target class. When querying the database for objects similar to a particular query object, the retrieval system ranks the objects in the database in descending order of similarity to the query. The SMEs then visually inspect the top k results. Due to resource constraints of the user, k is in the order of a few hundred results and practically never exceeds a tiny fraction of the total number of objects in the database. Often, these search results exhibit variations that are interesting variations of the original object of interest that warrant further investigation. These newly found objects are then themselves selected as queries for a subsequent search. This process of defining queries based on results of previous queries is common in information retrieval and is known exploratory search [185, 186].

The search is terminated once no more relevant objects are found upon subsequent search iterations. However, empirical studies show, that users are unable to effectively engage in this query-response process for an extended period of time [187]. This is, however, unavoidable due to the relatively long response time of the system and the need to manually verify the results (what is similar from a data-viewpoint may not be interesting to the user). This risks missing relevant objects by premature termination of the search. Broadly speaking, there are two ways to improve this process, namely (1) reducing the query response time of the system and (2) reducing the number of required interactions altogether [8]. Fast query-response cycles are extremely valuable in exploratory search [188]. The near-real-time response of web search engines like Google makes it possible for users to quickly narrow down their information need through a process of trial-and-error. Another way to avoid a prolonged query-response cycle is for the IR system to automatically refine the query so the search results coincide more closely with the search intent of the user. Various methods have been proposed to infer the user's search intent by caching the query history of previous search sessions, analyzing click-through rates [189] and building cognitive models of the user [190].

Reducing retrieval time There are technical limits to both the speed with which state-of-the-art search algorithms can rank the objects in a time series database and the extent to which these databases can be optimized to facilitate this scan. The upper bound in terms of runtime for ranking a time series database is a sequential scan of all objects. In a high-volume production environment (like the automobile industry) a typical search that considers data of a few months may encompass in the order of ten million objects. Assuming each object contains a thousand data points, a sequential scan of all objects under DTW would take around 18 minutes to complete on standard computing hardware [163] (a brute-force sequential scan would require an unfeasible 10^{12} operations to complete). This is much slower than the near-zero retrieval time of web search engines like Google that we are familiar with. This search speed is possible, because results are retrieved from indexed databases by matching a query with metadata tags of an object. Scanning of the content is not required *during* the search but essentially ”outsourced” to an indexing process. This process is a continuous background task that crawls the internet and attaches metadata to web pages. However, transferring this approach to time series databases is challenging. Multidimensional indexing structures that consider each point in time a separate dimension are slower than a brute-force sequential scan for long time series [104] and therefore unsuited. A widely adopted work-around is to extract features from the time series data and index this lower-dimensional space. This makes it possible to quickly retrieve the nearest neighbors to a search query in the feature space and calculate an exact solution using the raw data of the nearest objects. Naturally, the effectiveness of this approach fundamentally depends on the quality of the feature extraction [103]. To enable a high recall in the manufacturing context, these features need to accurately capture the presence of all possible fault patterns that span orders of magnitude in length. It was shown in section 2.3 that this is often not feasible in practice. Any index structure on a generic feature representation will improve system response time but severely limit the recall for some faults. This trade-off is not acceptable in practice. To ensure a high recall, there is often no way around a sequential scan of the database [191, 8].

Query adaptation based on relevance feedback Since speeding up the query-response cycles is not feasible, the focus must instead be shifted to reducing the number of interactions by optimizing the query. There are two ways to do this, namely (1) expanding the query using a set of predefined rules or (2) adapting the query based on relevance feedback of previous search results. The former is the standard approach for text-based queries but is not feasible for time

series data [117]. Thus, the only remaining option is to adapt the query based on relevance feedback. The idea is that the user provides feedback about the relevance of the search result to the system that adapts the query and retrieves a new result that is better aligned with the search objective. This has shown to be an effective way of increasing search recall for different applications [192, 193].

When the user submits one or multiple time series objects to the system, the IR system uses the optimized shapelet algorithm discussed in the previous section 4.4.1 to extract a set of query patterns from the objects. Thus, the system is able to simultaneously consider multiple patterns from different objects. The nearest neighbor in the database is found based on a majority vote of these patterns. This many-to-one nearest neighbor search has proven to be very robust in practice [8]. The objects in the database are ranked accordingly and presented to the user. As the user evaluates the results, the feedback can be used by the IR system to improve the query patterns and provide more relevant results. Depending on the data and the search objective, caching strategies that limit the re-ranking operation to the top-k results can achieve near-real time re-ranking of results.

In cases where the user wants to ensure that ”all needles in the haystack” have been found (such as when identifying safety-critical process faults), relying on a user-defined k is not acceptable. In these cases, the query-response cycle is rerun multiple times. As previously mentioned, this cycle time is in the order of minutes. The development focus was therefore laid on the reduction in the necessary number of these cycles, trading an increase in retrieval recall for a (considerably) longer runtime. The system achieves this by attempting to infer the feedback provided by the user and automating the query-response cycle without relying on manual input. This is known as pseudo relevance feedback (PRF) and used in other IR systems to increase search accuracy without requiring prolonged system interaction. The rationale of PRF is based on a fundamental hypothesis in IR known as the cluster hypothesis, whereby ”closely associated documents tend to be relevant to the same requests” [194]. The system implements this idea by searching the database using the result of a previous query, checking whether it is able to find a ground truth of manually annotated objects within the search results, accepting or rejecting the object accordingly and rerunning the query adaptation and search process [8]. The autonomous adaptation of the search query (set) as the search progresses is depicted in figure 32. In this particular case, the tightening tool slipped off the bolt during the tightening

operation due to misalignment between the tool and the fastener. Initially, two characteristic patterns are extracted and added to the initially empty query set. Both patterns describe a drop in the tightening torque that occurs during the slip-off of the tool. After a few iterations, the query refinement subroutine replaces one of these patterns with that of a steep increase in the tightening torque that occurs during tool re-entry of newly found objects. These two shapes in combination allow the algorithm to more easily find the next variation of the fault pattern, in which the tool skips one side of the hexagonal bolt head before re-entry. This example shows how the algorithm incorporates information from subsequent search results to discover variations that are increasingly different from the starting object. However, the PRF mechanism frequently gets trapped in single pattern variations (depending on the starting point) and generally suffers from a self-affirmation bias. For a more detailed discussion of the algorithmic design considerations and results, readers are referred to the original publication [8].

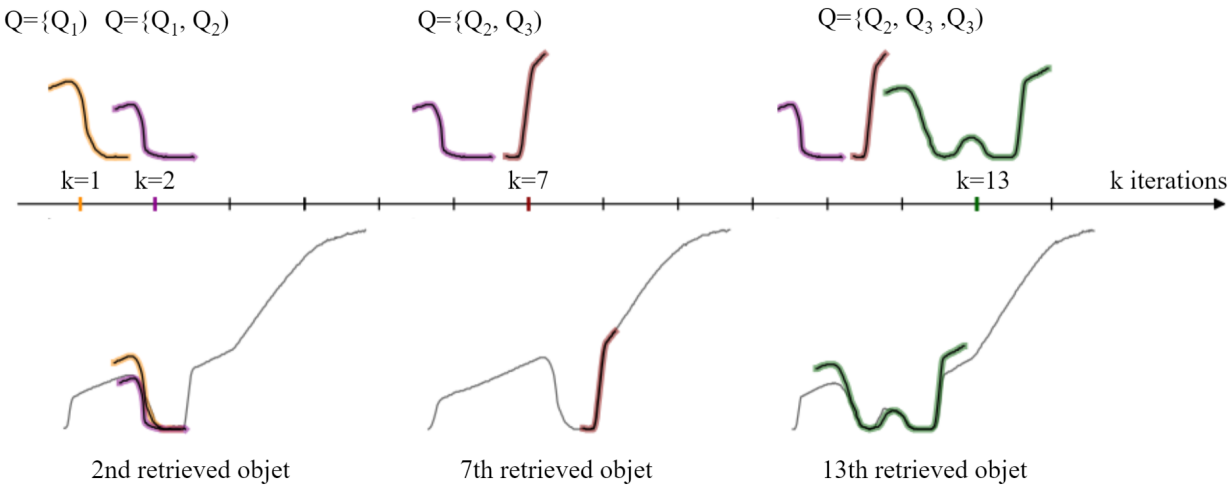


Figure 32: Changes to the query set of the adaptive search system over multiple relevance feedback cycles [8].

4.5 Continuous adaptation of anomaly detection models

The previous sections discussed the algorithms and systems that were developed to extract and refine patterns that are maximally characteristic of a specific class. This serves two distinct purposes that are briefly discussed in this section.

4.5.1 Adapting one-class models

The first (and arguably most important) purpose of these contributions is the ability to adapt the decision boundary of the one-class anomaly detection model - essentially providing a means for the model to ”remember” uncritical anomalies. This helps ensure the practical feasibility of the system, as repeated annotation of the same objects tie up resources and reduce the confidence of SMEs in the system. Given a number of objects that are detected as anomalies but uninteresting from a domain perspective, the first step is usually to use the IR system discussed in the previous section to increase the amount of available data. The input data set of the shapelet algorithm consists of (A) all available data of these uninteresting anomalies and (B) a representative subset of the ground truth of the normal majority class or, alternatively, a sufficiently large sample of unlabeled data. The latter is acceptable as long as the assumptions about the frequency of (interesting) anomalies discussed in the previous section holds. The pattern extraction subroutine of the shapelet algorithm is restricted to data set (A) and the patterns added to the library of the one-class anomaly detection model. This process is shown in figure 33.

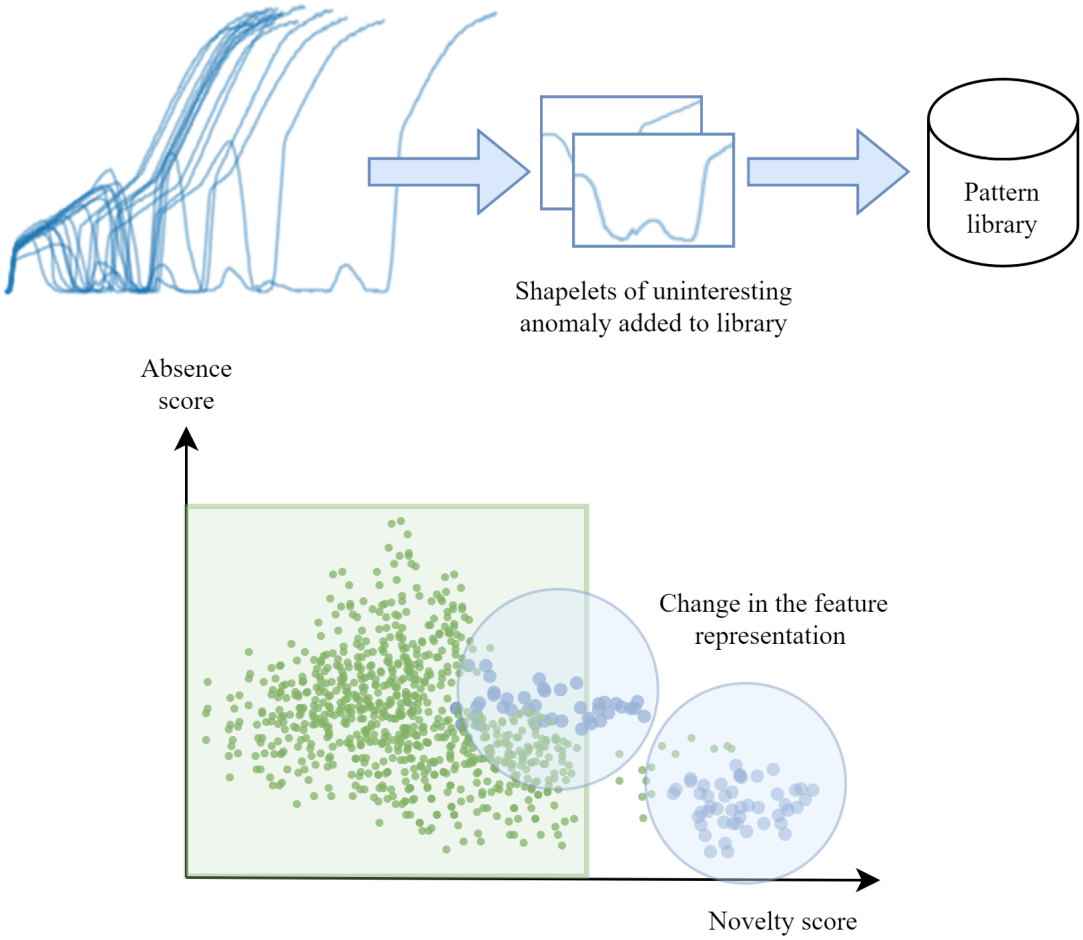


Figure 33: Objects are no longer detected as anomalies after their shapelets are added to the pattern library of the one-class model.

The advantage of the intuitive interpretability of these pattern-based models cannot be understated. Aside from model debugging, this helps SMEs consolidate and systemize their domain knowledge that has been fed into the system over time. A concrete real-world example is provided for a one-class model of a tightening process in figure 34.

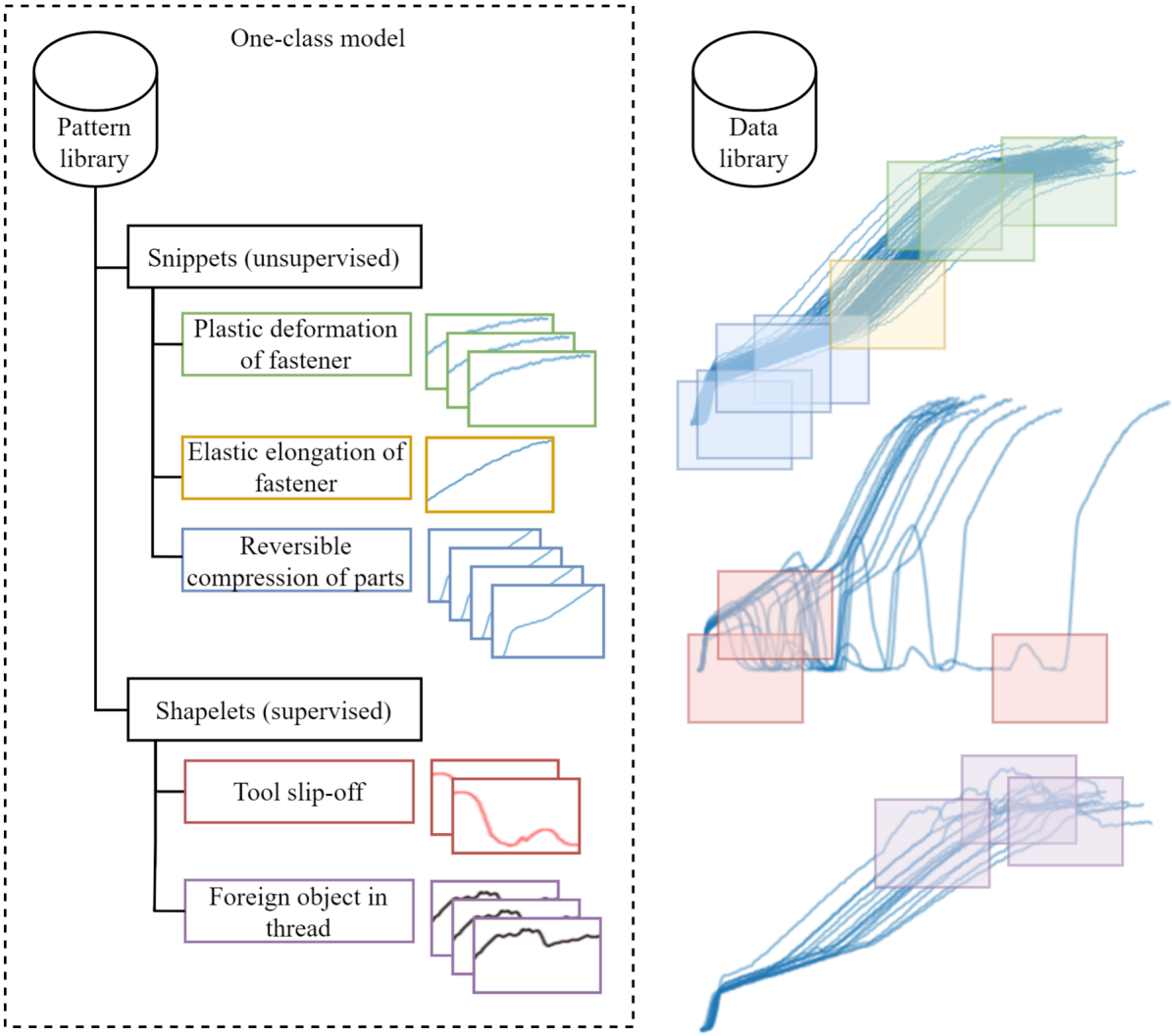


Figure 34: The extracted patterns can be annotated by SMEs to create a taxonomy of the domain knowledge learned by the model.

4.5.2 Training classification models

The main objective of the ML systems is to monitor a large number of manufacturing processes and detect anomalies that it does not recognize and therefore *might* indicate a process fault. In practice, once an anomaly has been detected and a fault confirmed, there are frequently calls by SMEs and decision-makers to deploy additional models to detect this specific *verified* faults. While countermeasures are underway, SMEs may have to rely on these models in the meantime. This requires a supervised classification model instead of the unsupervised one-class models that the majority of the system consists of. While this is a different model *type*, it turns out, that a suitable model can be constructed from the same . In this case, the shapelets are used to extract a shape-based feature representation of the data. The training data is compiled from annotated data of the target class and data from the majority class. Standard

ML algorithms are then used to learn univariate thresholds for each shapelet, resulting in an inherently interpretable classification model [178]. This approach to time series classification is essentially a two-step process of committing characteristic patterns to memory and learning to associate them with the underlying classes. The shapelet library, the matching subroutines and the classification model are packaged into a single model and stored in the model registry for deployment (cf. figure 16).

5 Deep learning for anomaly detection

5.1 Motivation for deep learning models

Many of the popularized achievements in the field of artificial intelligence, such as the defeat of world-class chess masters by Deep Blue's AI in the 1990s [195] and mastering the game of Go by DeepMind's AlphaGo Zero AI in the 2010s [196, 197], are attributable to the success of deep learning (DL). This technology has enabled numerous break-through developments for real-world applications in computer vision, speech recognition and language translation [198]. The last couple of years have seen an explosion in the application of deep learning to problems in different domains. This is in no small part due to powerful open-source machine learning frameworks like Google's TensorFlow [199] and PyTorch [200] (primarily invented by AI researchers from Facebook) [201]. Before discussing the motivation behind the use of deep learning for anomaly detection, it is helpful to define what deep learning is and how it relates to the broader fields of machine learning and artificial intelligence.

Definition 13 *Deep learning is a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data. (Oxford dictionary)*

DL is therefore a sub-field of ML, which, as discusses in the introduction, is itself a sub-field of what is broadly referred to as artificial intelligence. The relation between these three is schematically shown in figure 35.

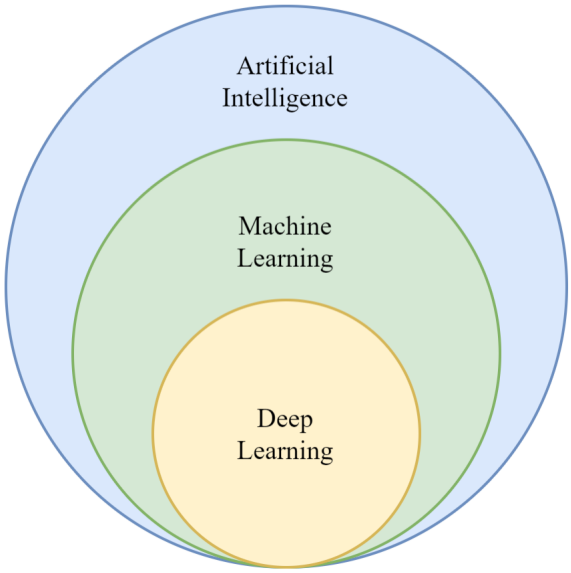


Figure 35: Deep Learning is a sub-field of machine learning, itself a sub-field of artificial intelligence.

At the heart of deep learning lie artificial neural networks (ANN). They are computational learning systems that learn to transform a data input into a desired output using a network of nodes [198]. At the most basic level, an ANN is an optimization algorithm used to approximate a mapping function between input and output [202]. The information processing and adaptation of this network is conceptually inspired by the neuron structure in biological brains. Multi-layer ANNs with more than one hidden layer are usually considered deep learning models, although there is no widely agreed definition. The trend to increase the number of layers came after studies in the 1980s showed, that model performance could be improved by increasing the number of layers [203]. Current state-of-the-art deep learning models for real-world applications are often very deep networks [197].

Why should a deep learning approach for anomaly detection in the manufacturing domain be considered? Based on the discussion of the challenges in section 2 and the review of existing literature in section 3, deep learning was deemed a promising approach for a number of reasons:

- There are theoretical and empirical studies that show, that ANNs learn to combine information from previous layers into increasingly abstract features. In principle, the ability of the model to learn hierarchical features directly from the time series data makes it possible to avoid the problem of information loss associated with the extraction of descriptive features discussed in section 2. Empirical results suggest that this is indeed the case. DL models regularly outperform conventional ML approaches for tasks such as time series

forecasting, classification and clustering.

- There is considerable research on DL algorithms for time series analysis generally, and anomaly detection in particular. The most competitive models for anomaly detection applications are based on ANNs [122] as ”deep learning completely surpasses traditional methods” [204]. These results have encouraged researchers to use DL models for various real-world anomaly detection applications. This is supported by the growing number of academic papers on applications of deep learning for anomaly detection in various domains [3]. The trend is shown in figure 36.
- The large amounts of data required for training DL models was readily available in the manufacturing setting.
- Many break-through results published at the time of this research employed DL models for problems that seemed much more sophisticated than the problem at hand.
- The user-friendliness of open-source DL frameworks in principle opened the possibility of high-level model training and management without requiring low-level debugging of the models.

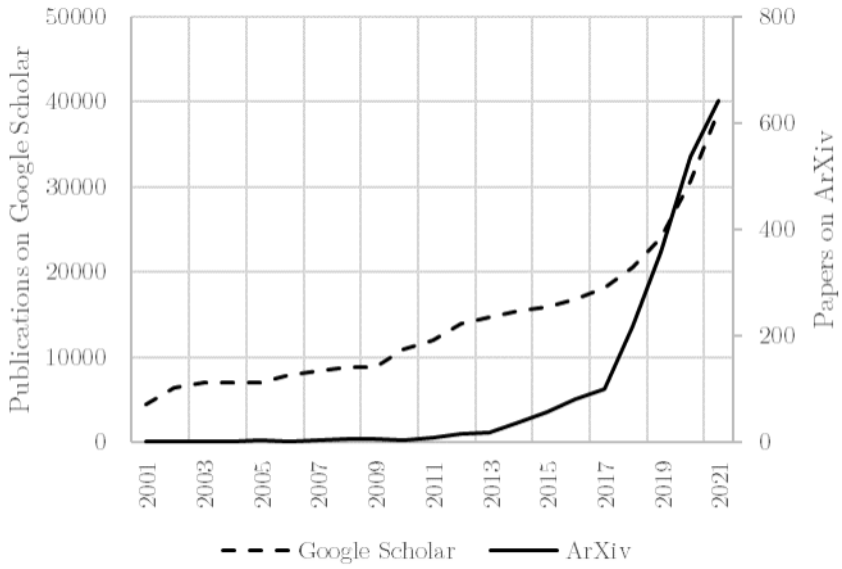


Figure 36: Number of academic papers indexed by Google Scholar and ArXiv that include the keywords ”deep learning” and ”anomaly detection”.

Despite the impressive results of DL for many applications, it is no silver bullet that can be thrown at any problem. Deploying this technology in production for anomaly detection comes

[3]The information was retrieved from the APIs of Google Scholar and ArXiv with the help of third-party code [205, 206]

with serious practical challenges. If retraining the model based on user feedback does not work (which is likely, given the extreme scarcity of annotated data) it is not immediately clear to data scientist what they need to do (point five above quickly turned out to be elusive in practice). How should they change the model? Simply ”making it deeper” will usually not solve the problem. Additionally, hyperparameter tuning is based on experimental results more than theory i.e. it requires a process of trial-and-error that does not scale well. The research summarized in the subsequent section was motivated by the goal of exploiting the feature-learning ability of ANNs without having to rely on extensive hyperparameter tuning. The idea was to develop an approach that would give up the notion of an end-to-end model for anomaly detection in return for a simpler and more robust model. This would make it possible for data scientists to train and manage a large number of DL models in production and was considered a decisive precondition for the scalability of a DL-based approach.

5.2 Fundamentals of deep learning

ANNs are computational learning systems, which refers to the neural network architecture itself as well as the optimization algorithms used to learn its parameters. To follow the design considerations and discussion in this section requires a high-level understanding of these systems. This section provides a brief overview of how ANNs process information, how they ”learn” from the data and the challenges of training deep networks, particularly for processing sequential data. Important concepts and implications are stated without mathematical proof - interested readers are referred to dedicated literature.

5.2.1 Multi-layer neural networks

An ANN is a network of interconnected nodes called artificial neurons, that are simplistic computational models of biological neurons [4]. To understand how ANNs process information, it helps to retrace the design motivations that went into the development of these neuron models. The first computational model of a neuron was a threshold logic unit. It was inspired by the way in which biological neurons process and transmit signals [209]. The dendrites of a neuron receive electric input signals from connected neurons that can be either excitatory or inhibitory. If the accumulated charge in the cell exceeds its activation potential, the neuron

[4]Over time, the development of these neuron models has moved away from biologically motivated design to improve empirical performance [207]. More recent approaches such as attention mechanisms and spiking ANNs are reversing this trend [208].

fires and transmits the signal to other connected neurons. Based on this understanding, the McCulloch-Pitts-neuron, named after its inventors, takes n binary inputs x and an inhibitory input i and outputs a binary output y according to the following Boolean function:

$$y = \begin{cases} 1 & if \sum_{k=1}^{n} x_k \geq \Theta and i = 0, \\ 0 & otherwise. \end{cases}$$

The neuron outputs a 1 if the inhibitory input i is inactive and the sum of its excitatory inputs x exceed the threshold Θ. This threshold can be set so that the MCP neuron performs different Boolean operations like logic gates.

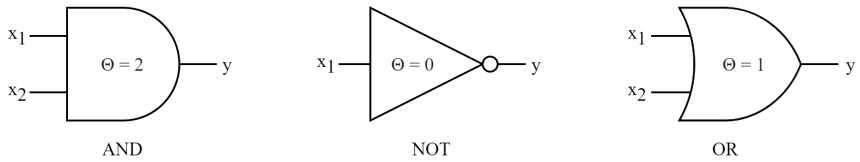


Figure 37: The MCP neuron can be used to implement different Boolean operations.

Based on this simple model, Frank Rosenblatt developed the perceptron in 1958 [210].

$$y = H(\mathrm{w} \cdot \mathrm{x} + b)$$

where w is a real-valued vector of weights, H is the Heaviside step function and b the bias of the model. This model can be visualized as a directed, weighted graph 38.

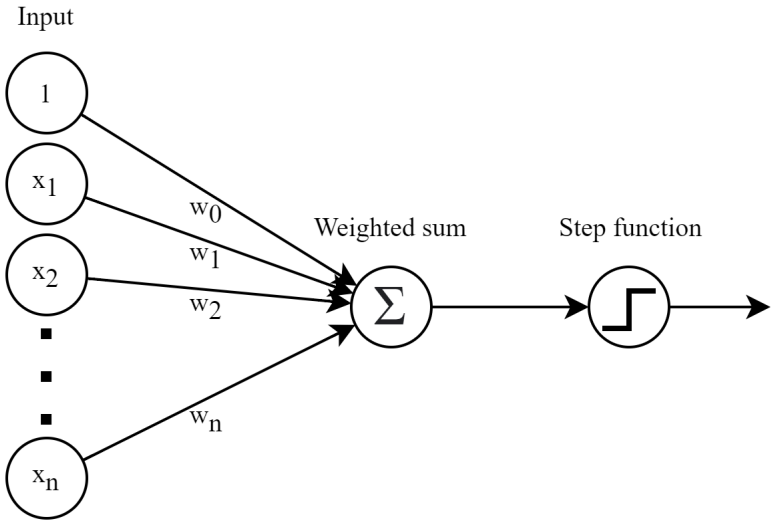


Figure 38: Graphical representation of the mapping function of the perceptron model.

Crucially, the perceptron is able to learn these weights w (as well as the threshold Θ) from

a labeled dataset using a simple supervised learning algorithm. The algorithm calculates the difference δ between the function output and the label and iteratively adjusts the weights until the algorithm converges.

Algorithm 1 The perceptron algorithm

while not converged **do**
$\delta = \hat{y} - \sigma(\mathrm{w}\mathrm{x} + b)$ ▷ Calculate difference
if $\delta != 0$ **then**
$\mathrm{w} = \mathrm{w} + \delta \cdot \mathrm{x}$ ▷ Change weights
$b = b + \delta$ ▷ Change bias
end if
end while

In short, the weights are adjusted proportional to the input that caused the error. The algorithm is guaranteed to converge if the labeled data is linearly separable [211]. Although groundbreaking at the time, the algorithm has considerable limitations regarding its ability to learn more complicated functions. It was shown early on, that learning even a simple non-linear function like the XOR function requires a more sophisticated algorithm [212]. A non-linear function is a function, that combines the inputs in a way that cannot be reproduced from a linear combination of the inputs. Graphically, this can be understood as the inability to separate two classes using a straight line on a two-dimensional plane as shown in figure 39. Since the decision boundaries of most real-world problems are non-linear, the perceptron is limited in its usefulness to solve relevant problems..

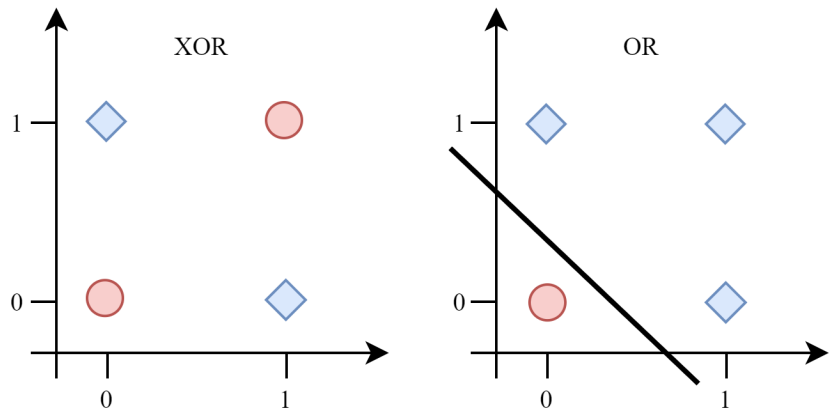


Figure 39: The linear OR-function (right) can separate two classes using a single line, while this is not possible for the non-linear XOR-function (left).

The seemingly straight-forward answer to this problem is to add additional layers between input and output as is shown in figure 40. Since a perceptron can model logic gates and a XOR-function can be constructed from these logic gates, a multi-layer perceptron (MLP) should be

able to model that function. The MLP is a type of (deep) ANN that is schematically depicted in figure 40 [5]. However, the perceptron algorithm cannot be used to train MLPs.

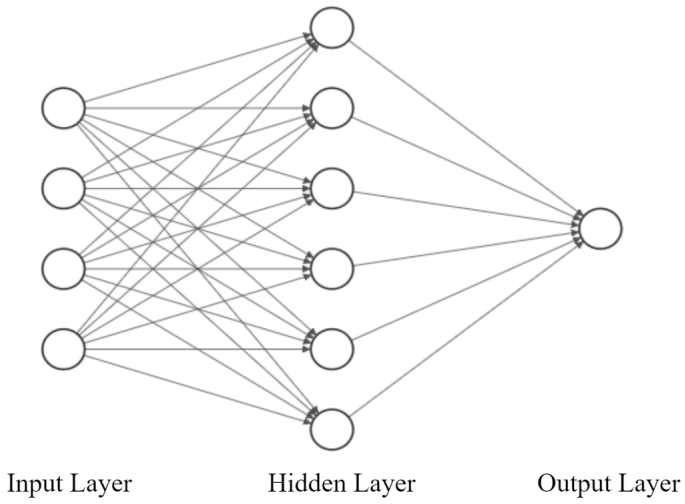


Figure 40: A multi-layer perception has one or more hidden layers between the input and output layers.

To train these networks, the learning task is formulated as an optimization problem [214]. The weights w and biases b are adjusted to minimize a cost function $C(\mathrm{w})$. This cost function measures the deviation between the desires output and the output of the ANN in response to an input x. Finding an analytical solution to this problem so that $\frac{dC}{d\mathrm{w}} = 0$ is intractable for larger networks. Instead, the weights are incrementally adjusted through a numerical optimization algorithm known as gradient descent. This algorithm relies on the gradient of the cost function with respect to the weights of the ANN $\frac{\delta C}{\delta \mathrm{w}}$. The algorithm that is commonly used to approximate this gradient is called backpropagation. To solve this optimization problem numerically requires replacing the Heaviside function for two reasons:

- Numerical optimizations algorithms are based on the idea of incrementally tweaking the input values of a function to change the output in a desired direction. Small changes in the weights and biases should result in small changes in the output. The Heaviside function returns only binary values which means the MLP does not exhibit this behavior.
- The backpropagation algorithm requires a differentiable activation function. The Heaviside function is non-differentiable at $x = 0$ and has a derivative of zero elsewhere. Without a gradient, the gradient decent algorithm cannot update the weights.

[5] Visualization based on open-sourced tool developed by Alex LeNeil [213]

The simplest function that meets these requirements is a linear function. However, this would mean that the output of a layer is simply a linear transformation of its inputs. Thus, any number of layers could be reduced to a single-layer model (avoiding this was the motivation behind an MLP in the first place). This is why ANN require non-linear activation functions to be able to approximate non-linear decision boundaries [6]. An ANN with just one hidden layer and a sigmoid activation function can already approximate any (continuous) non-linear function [202, 216]. There is no practical limit to the complexity of the mapping functions ANNs can learn. They can easily be trained to achieve zero training error on randomly labeled data [217] (provided a sufficiently large network). This means, that it can essentially remember input data perfectly. On the other hand, it can learn to construct any output from noise that is virtually indistinguishable from real data (this property is used by many generative deep models [218, 219]).

5.2.2 Backpropagation and gradient descent

This section provides a high-level explanation of the optimization algorithms used for training ANNs that define how these models ”learn” from data. This is important to understand the challenges associated with deep learning and unsupervised learning that are relevant for the later sections. The optimization algorithms iteratively adjust the weights and biases of the network to minimize the cost function $C(\mathrm{w}, \mathrm{b})$. The parameter change at each iteration is determined by the gradient of the cost function. The elements of the gradient vector quantify how much the cost will change due to a change in the weights and biases of the network. Mathematically, the gradient of a (multivariate) function is defined as the partial derivatives of the function with respect to its arguments. Calculating this gradient analytically is intractable for large networks. Instead, an algorithm to approximate the gradient, called backpropagation, is used. The process of iteratively adjusting the parameters in the direction of the (negative) gradient is called gradient decent. These two algorithms are the standard approach for training ANNs [220].

A rigorous mathematical proof of the backpropagation algorithm is avoided in favor of a short summary of the underlying intuition. The discussion is based on simple three-layer ANN such as the oen depicted in figure 40 and limited to the gradient of C with respect to the weights

[6] Strictly speaking, an MLP is a special *type* of ANN that uses threshold activations. Referring to ANN as MLP is a misnomer [215].

w. The gradient with respect to the bias follows the same logic. The activation a of a node in layer l is a (differentiable and non-linear) function of the weighted sum of the values of the nodes in the preceding layer $l-1$. This can be summarized as

$$a^l = \sigma(z^l) where z^l = w^l a^{l-1} + b^l \tag{1}$$

During the forward pass, the input data is processed by the network and mapped to the output vector a^3. The cost function is used to calculate the deviation of this output vector a^3 from the desired output y. The backpropagation algorithm calculates the partial derivative of the cost function C with respect to the weights of each layer. This process is recursively repeated, starting with the output layer. The partial derivative of the cost C with respect to the weights w^3 is the following:

$$\frac{\delta C}{\delta w^3} = \frac{\delta C}{\delta a^3}\frac{\delta a^3}{\delta z^3}\frac{\delta z^3}{\delta w^3} = 2(y - a^3)\sigma'(z^3)a^2 \tag{2}$$

The weights w^3 affects the input to the layer z^3 and therefore its final output a^3. Therefore, the partial derivative $\frac{\delta C}{\delta w^3}$ is expanded into three separate terms according to the chain rule. The first term is the change in the cost due to a change in the output activations. Since the activations are the only argument to the cost function, this is easy to calculate. Consider the mean squared error (MSE) as the cost function $C = (y - a^3)^2$. Then the derivative is $\frac{\delta C}{\delta a^3} = 2(y - a^3)$. The second term is the change in the activations due to a change in the input to the layer. This is simply the gradient of the activation function $\sigma'(z^3)$. The third term is the change in the input due to a change in the weights. According to equation 1, this derivative is equal to the activations of the previous layer. If a node in the the previous layer is only weakly activated, changing the corresponding weights will have little impact on the next layer. Consider the partial derivative of the cost with respect to the weights of the previous layer w^2:

$$\frac{\delta C}{\delta w^2} = \frac{\delta C}{\delta a^3}\frac{\delta a^3}{\delta z^3}\frac{\delta z^3}{\delta a^2}\frac{\delta a^2}{\delta z^2}\frac{\delta z^2}{\delta w^2} = 2(y - a^3)\sigma'(z^3)w^2\sigma'(z^2)a^1 \tag{3}$$

The first two terms describe the cost incurred in the output layer and are the same as in equation 2. This cost is passed to the previous layer by multiplying it with the weights w^2. The cost at each node is the sum of the errors of all connected nodes. Essentially, the cost is ”propagated” backwards through the network - hence the name of the backpropagation algorithm. The fourth and fifth term are analogous to equation 2. The backpropagation algorithm is essentially the chain rule applied recursively to each layer until the gradient in that layer can be described

as a function of the output y and the activations a of the intermediate layers. Thus, the algorithm uses only values that are already calculated during the forward-pass to approximate the gradient, making it highly efficient. After the gradient is calculated, it is possible to optimize the weights and biases using the gradient decent algorithm 2.

Algorithm 2 The gradient decent algorithm

while not converged **do**
$w = w - \epsilon \frac{\delta C}{\delta w}$ □Change weights
$b = b - \epsilon \frac{\delta C}{\delta b}$ □Change biases

end while

In practice, this algorithm is performed in a batch-wise process, where the average of the gradient of multiple samples is computed. This is referred to as *stochastic* gradient decent and helps reduce the computational time of the algorithm. Epsilon ϵ is called the learning rate, which defines how much the parameters are changed in the direction of the gradient at each iteration. The learning rate affects both the training time as well as the likelihood of the optimization getting trapped in a local minimum. For a non-convex cost function, points exist along the (multidimensional) surface defined by the function where the gradient is positive. These points are known as local minima. Most real-world problems are non-convex. The (stochastic) gradient descent algorithm is not guaranteed to find the global minimum of the cost function. Although this was long thought to be a critical problem, empirical evidence shows that the algorithm rarely gets stuck in local minima [221]. The reason for this is the large number of parameters of ANN. In the high-dimensional parameter space, most critical points are saddle points. To stay with the picture of a surface, the large number of parameters allow the model to ”slide down” from the saddle point along multiple dimensions. The large number of possibilities mean that the algorithm need not find the one optimal solution but can find one of many possible (nearly) optimal solutions. This explains why these models generalize so well [197, 198]

5.2.3 Training deep networks

The motivation behind increasing the number of layers is that it allows ANNs to learn increasingly complex features from the data [203]. However, increasing the number of layers in deep networks makes model training more challenging. The previous discussion of the gradient descent algorithm can help understand why:

- The backpropagation algorithm calculates the gradient of layer l based on the error propagated backward from the layer $l+1$. This error is scaled by the derivative of the activation function. If this derivative is smaller than one, the gradient decreases exponentially for each layer [7]. This is known as the vanishing gradient problem.
- The activations in layer l affect the gradient in layer $l+1$. Large activation values will lead to larger gradients in the subsequent layers that cascade through the network in a phenomenon known as exploding gradients.

These effects are particularly pronounced for deep networks, as a growing number of derivatives and activations are multiplied together. For decades, training of deep networks was all-together impossible due to exploding or vanishing gradients. The former can be managed by regularizing the model or limiting the gradients via clipping [222]. Vanishing gradients are generally more problematic. While a suitable choice of the activation functions can mitigate the problem, it does not resolve it. Training very deep networks was practically not possible until Hinton et al. proposed a greedy training algorithm "[trains] layers sequentially and greedily by tying the weights of unlearned layers" [223, 224]. Rather than training the network end-to-end, layers are trained successively one-by-one.

The problem of unstable gradients is particularly pronounced for recurrent neural network (RNN). RNNs are the go-to network architecture for sequential data, as it allows to process variable-length input data [225, 115]. Most ANNs require input data of fixed size, which may require prepossessing steps such as appending/dropping data points or mapping a sequence to a fixed length. This is often associated with a skew or loss in information. Thus, RNNs are central to the application of DL for anomaly detection in time series data. An RNN is a recursive network that stores a representation of the sequence in its "memory" called a hidden state h. The RNN takes as input a sequence $\mathrm{x} = (x_1, x_2, \ldots x_n)$ and recursively processes each element, updating the hidden state h_i at each step by mapping the input element x_i to the previous hidden state h_{i-1} using a function f_θ, where θ are the weights and biases learned by the model.

$$h_i = f_\theta(x_i, h_{i-1}) \tag{4}$$

A schematic network architecture of an RNN is depicted in figure 41. To apply the back-

[7] For this reason, activation functions usually do not have a gradient greater than one.

propagation algorithm for model training requires ”unrolling” the recursive network into a feed-forward network that is made up of a connected sequence of copies of the original network. Each network processes the hidden state of the previous network and a part of the sequence. This unrolled network is usually very deep. This by itself already makes training more challenging. Additionally, calculating the gradient at each layer requires repeatedly multiplying the *same* weight matrix with itself. This will tend to magnify disturbances which results in an intrinsically unstable situation. In practice, gated RNNs are used to avoid this instability. Widely known exmplaes include the Long Short-Term Memory model (LSTM), and gated recurrent units (GRU). These networks are based on more complex neuron models that are able to store information unchanged over multiple inputs. An in-depth explanation is not necessary to understand the approach presented in this section - readers are instead referred to pertinent literature [226, 227].

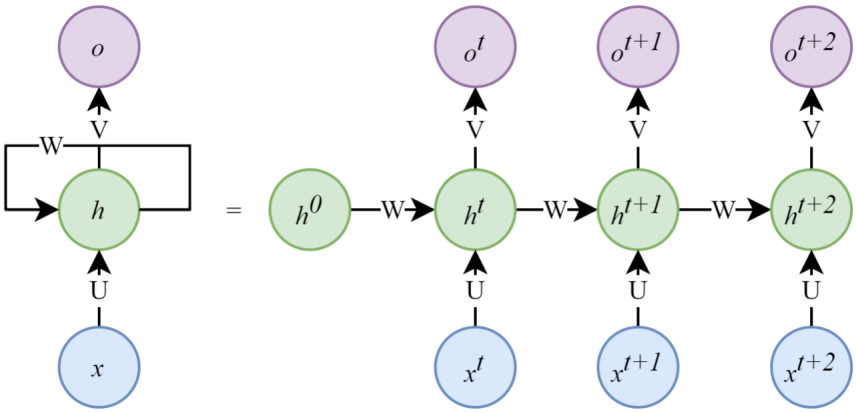


Figure 41: The recursive network of an RNN is ”unrolled” to train the model using backpropagation ”through time”.

5.3 Semi-supervised anomaly detection

This section proposes a semi-supervised ANN model for anomaly detection. The training approach is semi-supervised since the model makes assumptions about the unlabeled training data. The following subsections discuss various design considerations regarding the network architecture that motivated the final design choices of the model. Note, that the final model constitutes one *possible* way to implement these considerations. The abundance of hyperparameters of ANNs and the speed of development in the field means that vastly different approaches are possible, even if the same considerations are taken into account. While some elements have been established as de facto standards for specific applications (e.g. convolutional neural

networks (CNN) for machine vision and LSTMs for natural language processing), a plethora of papers continue to be published that propose new architectures for specific tasks. Most of these can be considered minor variations or extensions of standard architectures. The aim of the research described in this section is not to add ”yet another one” to this list, but to show how DL *can* be applied for anomaly detection in the manufacturing domain. As discussed in section 2, DL models have so far seen very little large-scale productive deployment in manufacturing settings.

5.3.1 Recognizing patterns in time series data

The first consideration is to ensure the ability of the model to detect patterns in time series data. As discussed in section 2, one of the challenges of analyzing time series data is that the value of a data point is usually meaningless without its temporal context. Thus, the model must analyze contiguous sections of the sequence at a time to detect structures and patterns in the data. This is similar to analyzing written text. Individual letters only convey meaning when strung together to form words, which must themselves be organized into coherent sentences. The answer to this problem is to focus the information processing of the network on one subsection at a time in order to learn coherent patterns. This is known as a convolution operation. It has been used in the field of machine vision to address the issue of pattern recognition for image data [8]. The same approach is adopted for processing time series data. A convolution operation can be interpreted as measuring the similarity between a section of a (larger) input vector and a (shorter) vector known as a kernel. For sequence data, this means comparing a shorter subsequence to a longer input sequence. Note the similarity to the sequence matching subroutines discussed in section 4.2.1. The only differences is that a convolution operation compares these patterns by a mathematical convolution operation (hence its name). For real-valued data, this is the same as the cross-correlation which is the preferred implementation in popular ML frameworks as it avoids the expensive integral calculation required for a ”real” convolution [199]. Calculating the cross-correlation between a subsequence and a longer sequence is known as a sliding dot product and is a common approach to search for a known feature in long signals in signal processing.

Just as in the case of the similarity join, the question of an appropriate kernel size needs to be

[8] The development of the convolutional neural network (CNN) by LeCun et al. at Bell Labs in 1989 [203] jump-started the development of deep learning.

answered. The standard approach is to try out different kernel sizes similar to the trial-and-error search of different subsequence lengths. However, this process of hyperparameter tuning is difficult to automate for unlabeled data. To allow for an automated training process, the same data is processed in parallel by multiple kernels of different sizes instead of relying a single kernel size. The idea is to have the network learn which of these sizes are best-suited to accurately detect a pattern and ignore those that are not. To combine these separate information streams, the hyperparameters of the parallel convolutional layers (padding, stride, dilation, etc.) can be adjusted so that their outputs are the same size and can be easily aggregated. This layered processing of the input data using different kernel sizes is not a novel idea. It is central to state-of-the-art deep CNN for image recognition [9]. This concept is illustrated in figure 42.

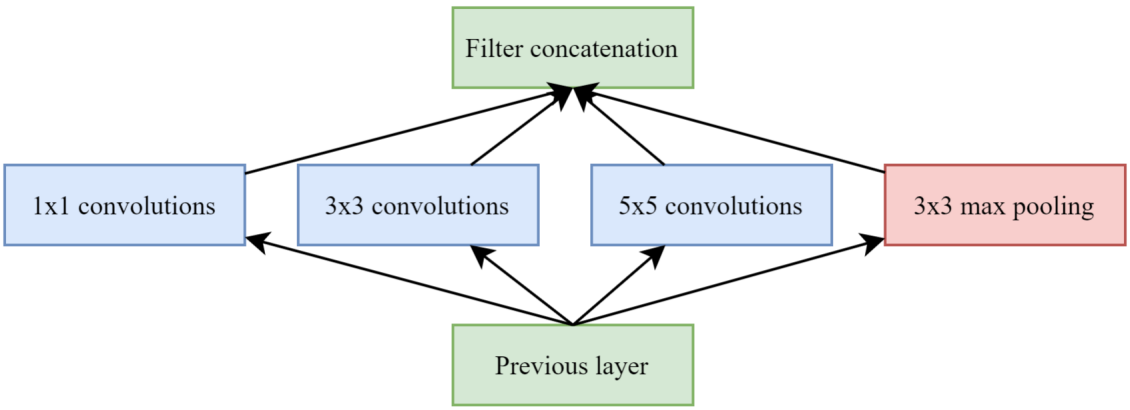


Figure 42: An inception module proposed by [9] that is used in deep CNN for machine vision applications.

So far, the only conceptual difference between how this ANN and the data mining algorithms described in section 4.3 analyze a time series is that the kernels are ”learned” from the data rather than extracted directly. The data mining algorithms extract template patterns from the data through an optimized trial-and-error search whereas the neural network learns prototype patterns by minimizing a loss function. Aside from the ability to detect the presence/absence of patterns of various lengths, the model must consider the *position* of those patterns along the time series. This is accomplished by using an RNN that processes the data sequentially. As subsequent sections of the time series are fed to the RNN, the model stores the information of what is has processed so far in its hidden state. A pattern that has been shifted in time results in a changed hidden state. Thus, the RNN encodes information on the position of its input patterns.

[9] In 2014 Szegedy et al. proposed a neural architecture called *inception* that beat the previous record in the famous VisionNet image recognition challenge [9]

5.3.2 Anomaly detection using latent variables

Motivating the use of latent variables Many reported applications of deep learning for anomaly detection rely on the so-called autoencoder model as a DL model and the reconstruction error as an anomaly score [115, 113, 117, 228]. The learning objective of the autoencoder is to (1) map the input data to a feature space and (2) reconstruct the input data based on the feature representation of the input data in that same feature space. The difference between the original input and reconstructed output is called the reconstruction error or prediction error (depending on the model). There are two reasons why this approach to anomaly detection is so popular: first, it is easy to implement. Autoencoders are one of only a few unsupervised models, they are simple to construct and the reconstruction error used for anomaly detection is calculated anyways as part of the loss function for model training. Secondly, the underlying idea seems plausible: the model learns to represent data in a feature space from which it is able to reconstruct that same data. Thus, these features must contain all information that characterizes the input data. If a large amount of unlabeled process data is used to train the autoencoder, it would seem reasonable to assume that it has learned the most important features characteristic of the majority normal class. If the model fails to reconstruct an input sequence using the same feature extraction logic learned from the training data, then the input data must contain (anomalous) features it has not seen before. While this approach to anomaly detection has been successfully applied to numerous real-world applications, its drawbacks become apparent during large-scale maintenance of many of these models in production. The problem is, that the use of a single reconstruction-based score results in a very high-level approach to model management - either the score is sufficient to detect anomalies or it is not. While initial unsupervised model training and deployment is straight-forward and the models tended to perform well as simple anomaly detectors [117, 229], model adaptation proved much more difficult. Due to the black-box nature of the model, the only option of the ML practitioner to retrain the model based on user feedback is to revert to trial-end-error hyperparameter-tuning. Attempts to develop a systematic process for adapting these autoencoder models over numerous field-testing cycles failed. The effort required for managing a large number of models was not practicaly feasible.

Instead, the idea is to pursue a more low-level approach that makes direct use of the features the model has learned. Rather than making assumptions about how well the input sequence conforms to the rules the model has learned to abstract and reconstruct the input data, this

feature representation is analyzed directly. Essentially, this is like asking the question directly ”Does the input data contain anomalous features?” rather than inferring an answer ”If the output of the model output is different from the input, then the input must contain anomalous features”. The intermediate layers of the DL model are often referred to as latent variables. This term is borrowed from the field of statistics to describe variables that are not observed but rather inferred indirectly by a (statistical) model. This naming convention is adopted in this section. The use of latent variables for anomaly detection provides two benefits for data scientists:

- It facilitates model debugging by helping understand why a particular object was detected as anomalous (or not). Correlating individual latent variables with observable features in the data may provide clues as to what the model has learned, making it is possible to leverage their feature-learning ability. Investigative studies from the NLP field show, show that this is indeed possible [118, 119, 230]. Using models trained for text translation, researchers could show that some latent nodes corresponded to interpretable features such as keeping track of quotations [230].
- It makes it possible to find a more complex and often better-suited decision boundary. As discussed in section 2, anomaly detection is inherently a threshold problem. A latent variable model for anomaly detection can use different thresholds for different variables, instead of relying on a single threshold for the reconstruction error. During model re-training, it may be sufficient to adapt only these thresholds without having to change the model’s weights and biases. Changing any of the network parameters changes the entire processing logic of the model. This is known as the CACE-principle (”changing anything changes everything”) [36]. This should be avoided if possible, as it requires a lengthy model training and validation process to ensure that previously learned structures were not ”forgotten”. This issue is discussed in detail in section 5.4.

Resolving the threshold problem Recall, that an ANN is an algorithm to approximate a mapping function from one vector space to another. In a deep network, the input data is successively transformed between vector spaces as it passes through its hidden layers. The representation of the data in these intermediate spaces reflects what the ANN has ”learned” from the data during the training process. To be precise, the ANN learns to adjust its *weights* i.e. the mapping *between* these vector spaces. While each layer in an ANN constitutes its own latent vector space, the analysis is made easier if the dimensionality of the latent space

i.e. the number of latent variables is small. Since the information processed by the model flows through each layer (in conventional feed-forward networks), the model must compress all information into this low-dimensional space. In principle, the input data may take on any arbitrary representation in this latent space as long as it helps minimize the cost function. If the learning objective is merely to minimize the reconstruction error of the model (as is often the case in literature), the latent variables may take on any arbitrarily complex distribution. This means, that it is not possible to decide whether a point in the latent space is anomalous by itself. Instead, it is necessary to evaluate the point relative to the latent representations of the majority class. Essentially, this requires defining clusters of normal points in the latent space, as depicted in the first subplot of figure 43.

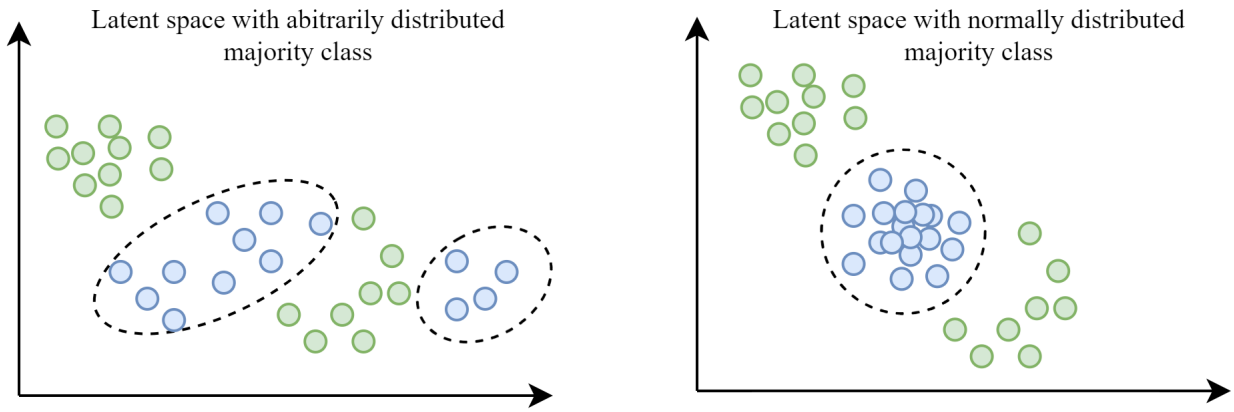


Figure 43: Schematic representation of the latent space of most conventional DL models (left) and with a normally distributed majority class (right).

This opens up a number of challenges associated with clustering multivariate data. The literature provides a host of probabilistic, partitioning and density-based clustering methods. However, the effectiveness of clustering algorithms critically depends on the choice of the hyperparameters of the algorithm. The difficulty of automatically setting these hyperparameters for real-world data has been thoroughly documented [231, 75, 117]. Thus, that in addition to training DL models, data scientists would have to deal with the challenges associated with clustering data in high-dimensional space. The need for clustering the data could be avoided, if the latent variables conformed to a predefined distribution. This can be achieved by changing the loss function of the model so that it learns to map the input data accordingly. What distribution of latent variables would make sense? For anomaly detection applications, the model should learn latent features characteristic of the normal class. If we assume these features to represent patterns in the process data that are themselves caused by an underlying physical process, and assume those processes to be normally distributed (as many natural phenomena

are), then it would makes sense for the latent variables to be normally distributed. This allows to use simple statistical thresholds, greatly simplifying the threshold problem and making it possible to automate the unsupervised learning process.

Training variational autoencoders As discussed, the ANN should map input data x to the latent space in such a way, that the latent variables z are normally distributed. For the latent variables to conform to a predefined normal distribution, a suitable term must be added to the cost function. This term must quantify the difference between the learned distribution of the latent variables $p(\mathrm{z})$ and the desired normal distribution $N(\mathrm{z})$. The learned distribution is often called the posterior distribution and the predefined (normal) distribution the prior distribution. In 2013, Kingma et al. [232] proposed an ANN called variational autoencoders (VAE) that implements this cost function. The VAE is a generative latent variable model that, in addition to ensuring the normal distribution of the latent variables, esnures that the latent space is continuous i.e. the mapping between the input layer and the latent layer is a continuous function. This means that a small variation in the input data causes a small variation in the latent feature representation. This is schematically depicted in figure 44. Thus, objects that are close together in the input space will be close together in the latent space. This makes interpreting the latent space easier, as data with characteristic features will tend to cluster together.

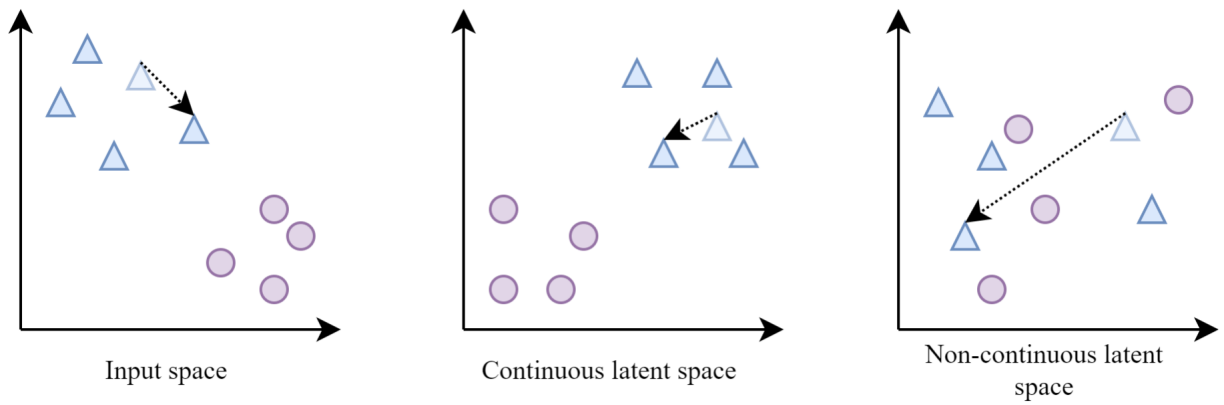


Figure 44: A continuous latent space is the result of a continuous mapping function which results in similar representations being conserved across spaces.

A normal distribution is defined by its mean μ and its standard deviation σ. Instead of mapping the input x to a latent vector z it is mapped to *two* layers, namely a mean vector $\mu(\mathrm{x})$ and a standard deviation vector $\sigma(\mathrm{x})$. An exponential activation is added to $\sigma(\mathrm{x})$ to ensure that it is positive (there is no negative standard deviation). Note, that a truly multivariate model of the

latent space would require a covariance matrix that describes how each variable is correlated to each other variable. However, VAEs only constrain the univariate distributions i.e. consider only the diagonal values of the covariance matrix. To compare the distribution of the prior and the posterior, points from the posterior distribution are *sampled* (to be precise, points are sampled from a normal distribution parameterized with the learned values for μ and σ) and compared to points sampled from a prior normal distribution of mean $\mu = 0$ and standard deviation $\sigma = 1$. Two distributions can be compared by calculating the statistical distance between them, referred to as a divergence. The Kullback-Leibler divergence is used as a loss term. It can be interpreted as "drawing out the two distributions, and wherever they do not overlap will be an area proportional to the KL divergence." [233].

$$KL(p(z|x)||N(z)) \tag{5}$$

The process of randomly sampling data points from the posterior distribution is non-deterministic. This is a problem for model training. Recall, that the backpropagation algorithm is used to calculate the gradient of each parameter θ with respect to the final output. It is not possible to calculate a gradient for a stochastic process. To make it possible to use the backpropagation algorithm, the stochastic nature of the sampling process is separated from the deterministic *learnable* parameters. Instead of randomly sampling the latent vector z from a distribution parameterized with the learned values for μ and σ, its value is determined by a function that takes as an input the mean μ, variance σ and a stochastic variable ϵ sampled from a normal distribution. As a result, the gradient between z and μ and σ can be calculated. This is known as the "reparametrization trick" and depicted in figure 45

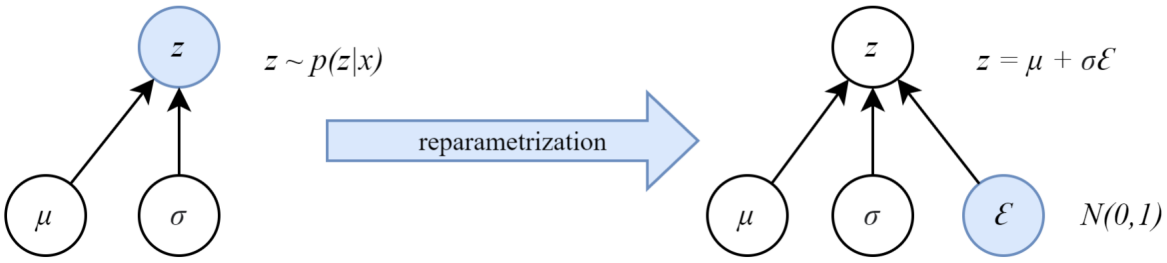


Figure 45: The reparametrization trick makes it possible to learn deterministic values Θ for a probabilistic latent vector z

The latent variable model (LVM) for anomaly detection proposed in the next section adopts the formulation of the KL-divergence loss and the reparametrization trick for learning a normal distribution of the latent variables. It does not, however, make use of the reconstruction error

of the decoder, as is typically the case in VAEs. There are two reasons for this. First, VAEs are typically used as generative models to create objects that are similar to the dataset it was trained on. For anomaly detection, there is no such need to generate new objects. Second, VAEs suffer from a well-documented phenomenon known as posterior collapse, where the output becomes disconnected from the input x [234, 235]. This is schematically shown in figure 46. This phenomenon occurs due to one of the following reasons:

- The signal from the input x to the posterior is too weak, which results in μ and σ approaching constant values that are irrespective of x. The posterior distribution essentially becomes disconnected from x. Thus, the decoder receives a constant signal $q(z|x) \approx q(z)$ for different x.
- The signal from the input x to the posterior is too noisy, which results in noisy μ and σ. To reconstruct a sensible $\hat{x}$, the decoder learns to ignore z. Thus, z becomes disconnected from $\hat{x}$.

Papers continue to be published that investigate the issue of posterior collapse [235]. The main advice is to ”weaken” the decoder to enforce a stronger dependency between the latent variables and the output. The proposed model avoids this issue by removing the decoder entirely.

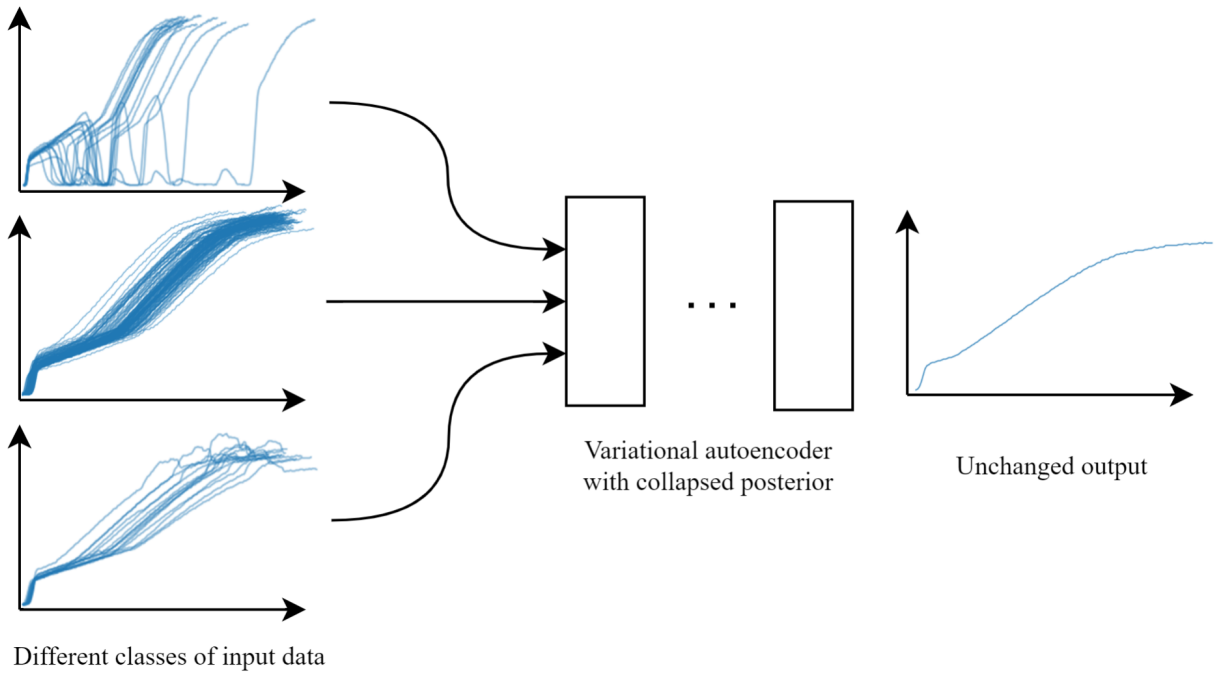


Figure 46: A VAE with a collapsed posterior will reconstruct the same output y for different inputs x

To summarize, enforcing a normal distribution of the latent variables has the following advantages:

- It avoids the clustering problem and allows to define simple statistical thresholds that correspond to an intuitive understanding of normality.
- It allows to quantify the uncertainty of the model associated with a particular point belonging to the normal class. This allows to prioritize the labeling of a given number anomalies.
- It acts as a natural regularization. Empirical studies have shown, that regularization improves the ability of ANNs to detect meaningful patterns in the data.
- It has been shown to enforce a weak form of feature disentanglement. This means that the latent variables will capture more diverse features of the input data which should allow the model to detect a broader range of anomalies. Additionally, feature entanglement is the main reasons neural networks are so difficult to interpret [83]. Thus, it will tend to improve the interpretability of the latent layer.

5.3.3 Latent variable model for anomaly detection

Model training and anomaly detection This section proposes an LVM for anomaly detection. The model design was motivated by the theoretical considerations discussed in the previous section as well as practical challenges encountered during productive deployment of the model. To enable training and deploying a large number of these models, data scientists must be able to orchestrate the training process. This section (1) presents a high-level architecture of the model (2) explains how it is trained in an unsupervised setting and (3) demonstrates its ability to detect anomalies in both real-world manufacturing data as well as benchmark data. The latter is particularly important to show that the proposed model is not fine-tuned to the particular setting but generalizes to other applications.

The LVM combines two network architectures: a CNN inception network and a (gated) RNN that serve two distinct purposes. First, the input data is passed to the inception network, that uses parallel kernels of different size to recognize contiguous patterns in the time series. Each kernel is able to detect features of different lengths. The output of the inception network is a feature map, that encodes information about the presence of those features in the input data.

The inception network serves as the model's *feature detector*. This information is passed to the RNN, that analyzes the presence and order of these features. The RNN scans across the feature map, sequentially processing the strength and order of the feature activations. This scan is similar to the sliding window used for subsequence matching of time series patterns discussion in section 4. If the input to the RNN is different from what it has previously learned from the data, its final hidden state will be different as well. Thus, the RNN essentially serves as a *feature analyzer*. This hidden state is mapped onto a set of latent variables using a mapping layer. The location of the point in the latent space determines the anomaly score of the object. The mapping layer and the thresholds applied to the latent variables together form the *anomaly detector* of the system. This architecture is summarized in figure 47, where the information flow during the forward pass of the model is depicted using black arrows.

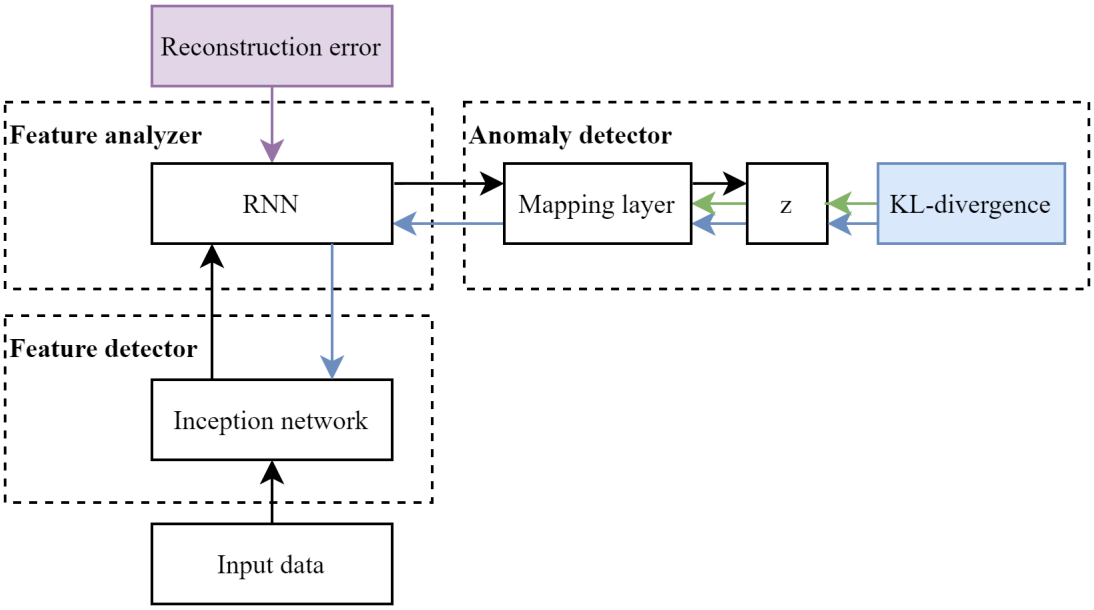


Figure 47: High-level architecture of the latent variable model for semi-supervised anomaly detection.

The cost function for unsupervised model training consists of two separate terms. The first term is the reconstruction error between the output of the RNN and the input data. This term ensures, that the RNN is able to reconstruct the input data from the feature map i.e. that it is able to interpret the feature map. This cost is passed backward to train both the RNN and the inception network. This does not risk the collapse of the posterior z as it is applied *before* the latent layer. The second term is the KL-divergence calculated from the distribution of the latent variables in the final layer. This loss is passed through the entire model and used to train the mapping layer as well as the RNN and inception network. The combined cost function is

provided below:

$$L_{unsupervised} = L_{reconstruction} + L_{KL-divergence} \tag{6}$$

These two loss terms are trained in *alternating succession* rather than simultaneously. As discussed in section 5.2.3, training layers sequentially helps avoid the vanishing gradient problem in deep networks. First, the model is trained using only the first term $L_{reconstruction}$. This ensures, that the inception network and the RNN are able to recognize and reconstruct features of the training data. The anomaly detector is not trained, as no loss is propagated backward. In a second step, the model is trained using the second term $L_{KL-divergence}$, whereby the weights of the inception network and the RNN are frozen. Thus, the mapping layer learns to map the hidden state of the RNN to a normally distributed latent space. In a final step, the layers are unfrozen and the combined cost function $L_{unsupervised}$ used to fine-tune the model. If the model does not converge, the first two steps are repeated in alternating order. The training process can be automated without requiring user input. The flow of information of the backpropagation during model training is depicted in red in figure 47. Note, that there are two mutually exclusive paths for backpropagating the loss to the mapping layer. One is used for unsupervised training model training, while the other is used for the continuous adaptation of the model. This is discussed in the next section.

Theoretical considerations aside, how well does this model perform in the field? How well does it generalize to different processes and to what degree does it lend itself to automated training and easy maintenance? To answer these questions, the model underwent extensive field-testing before it was deployed and assessed at scale (discussed in the next section). To make it easier to compare the results the pattern matching model discussed in section 4, the same tightening process data was selected for model training. The distribution of the latent variables is shown in figure 48. The latent features of the majority of input objects are normally distributed around the same mean $\mu = 0$. The annotated anomalies are often clustered around the tails of the distribution and are even linearly separable for some variables. An important thing to note, is that different anomaly classes take on similarly different values for the latent variables. This suggests, that the model is indeed able to learn distinct features from the input data. For the sake of better interpretability of the latent space, a principal component analysis is used to create a linear two-dimensional projection of the data. Using this projection, both anomalies

can be linearly separated from the majority class using a single threshold. Since the principal components are a linear combination of the latent variables, these anomalies can be detected by applying *linear* thresholds on these latent variables. This makes it relatively easy for data scientists to retrace model decisions (the same visualizations shown in figure 48 were repeatedly used by data scientists during the deployment phase). Note, that while this ability to linearly separate anomalies based on their latent representation is desirable, it need not be the case.

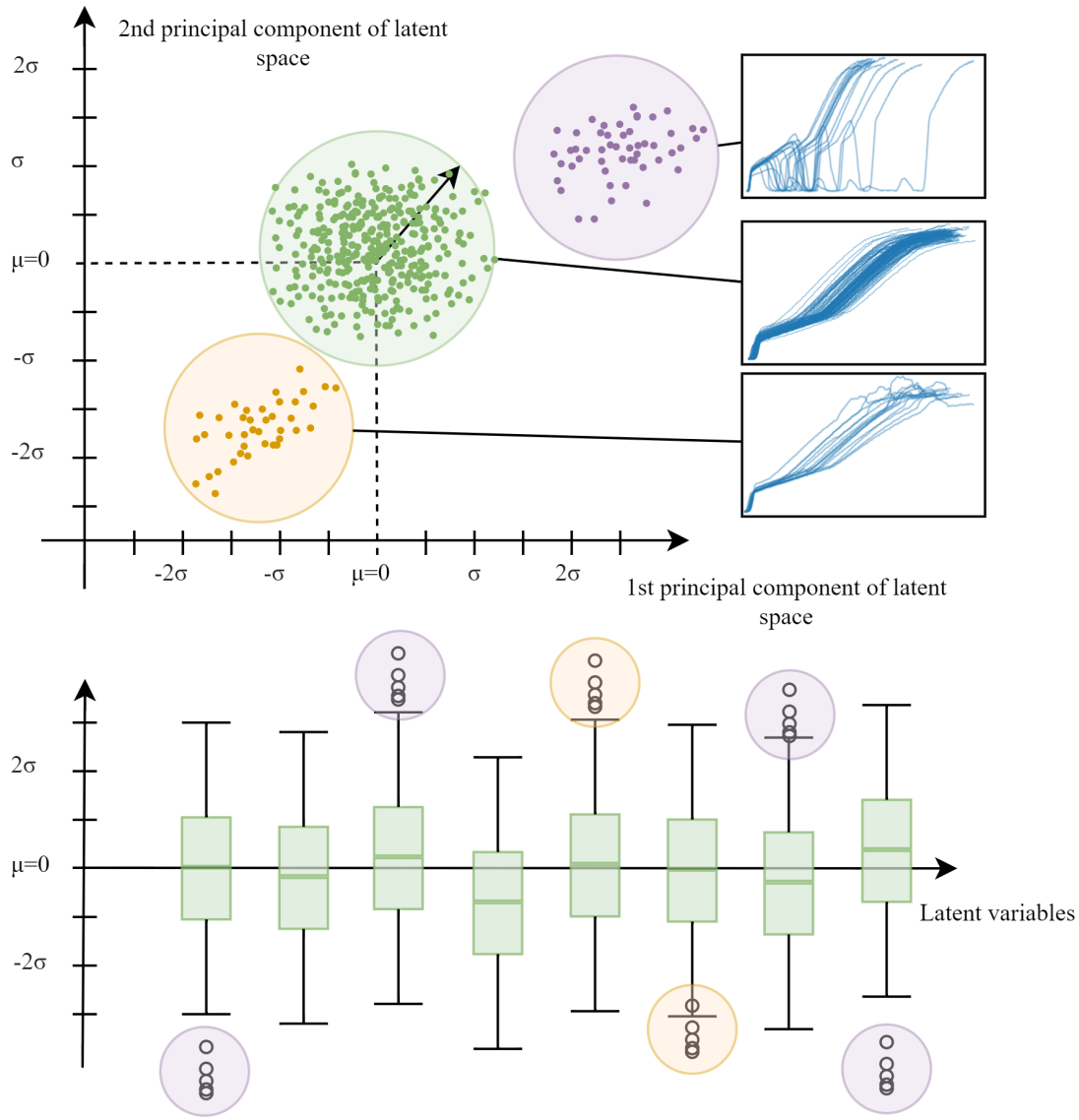


Figure 48: The model detects different features in the latent space for visibly different input data.

This example shows, that the proposed LVM *can* be used for anomaly detection in tightening process data for a particular manufacturing setting. However, it is important to evaluate to what extent this approach is applicable to other processes in different settings. Otherwise, we have simply created ”yet another model”. While the considerations in section 5.3.2 are generic

and not specific to the particular setting at hand, it would be disingenuous to suggest that the approach generalizes to other settings without proper validation. Thus, it was applied to public benchmark dataset as was done for the pattern extraction algorithm in section 4.4.1. To validate the LVM the MNIST image dataset was used. This dataset consists of approximately 70.000 images of handwritten digits and is one of the most widely used benchmark datasets in the field of Machine Learning [236]. Although the model is intended for time series analysis, there is nothing stopping us from applying it to image data by simply flattening it into a one-dimensional vector (the same format as time series data). This is the standard method of processing image data for computer vision. Since the dataset consists of ten classes of (roughly) equal size, it is not possible to randomly sample the data for training as was done with the unlabeled process data. Instead, the model is trained using data of only one (randomly chosen) digit. Essentially, this digit is the ”normal majority class” and all other digits should be detected as anomalies during testing. Figure 49 shows that this is indeed the case. The latent representation of the digit one is normally distributed and different digits form their own clusters towards the tail of the distribution. This shows, that the model can be generalized to data other than the process data of the particular manufacturing setting for which it was developed.

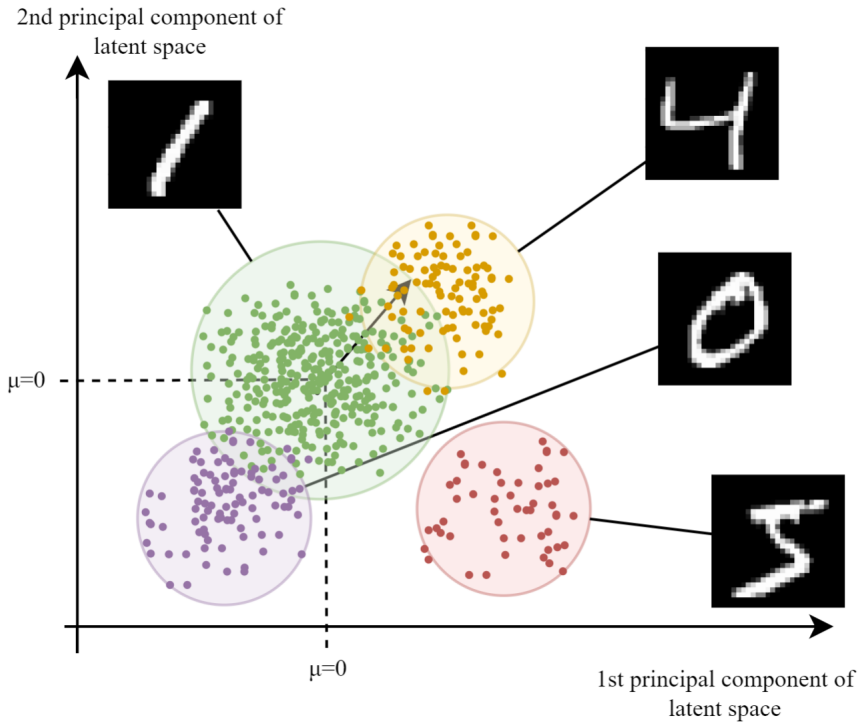


Figure 49: A linear two-dimensional projection of the latent space of the LVM that was trained using images of the digit one of the MNIST dataset.

Correlating input features with latent variables As discussed in the previous section, there are two motivations for using an LVM. First, it allows the model to use simple thresholds that can be set automatically (provided the variables are normally distributed). Second, it give data scientists some ability to analyze what the model has learned. This makes it possible to ”look inside” a model that are usually accepted to be black boxes. For most variables, no obvious correlation could be established, even if they were useful for detecting anomalies. A few, however, did show a (weak) correlation with visible feature in the input data. Figure 50 shows an example of selected latent variables that were correlated with a different duration of the clamping phase and the presence of a ”bump” in the curve. These results are similar to the results reported by NLP researchers [230]. There is, of course, a lot of uncertainty associated with identifying such correlations - they are identified by manually comparing the input data and the distributions of the latent variables. Data scientist may wrongly focus their attention on spurious patterns and miss others (especially multivariate correlations that are difficult to detect). However, once a solid correlation is established, data scientists can manually adjust the threshold of that variable to alter the classification decisions of the model. This is very similar to the snippet model for anomaly detection discussed in section 4.2.3.

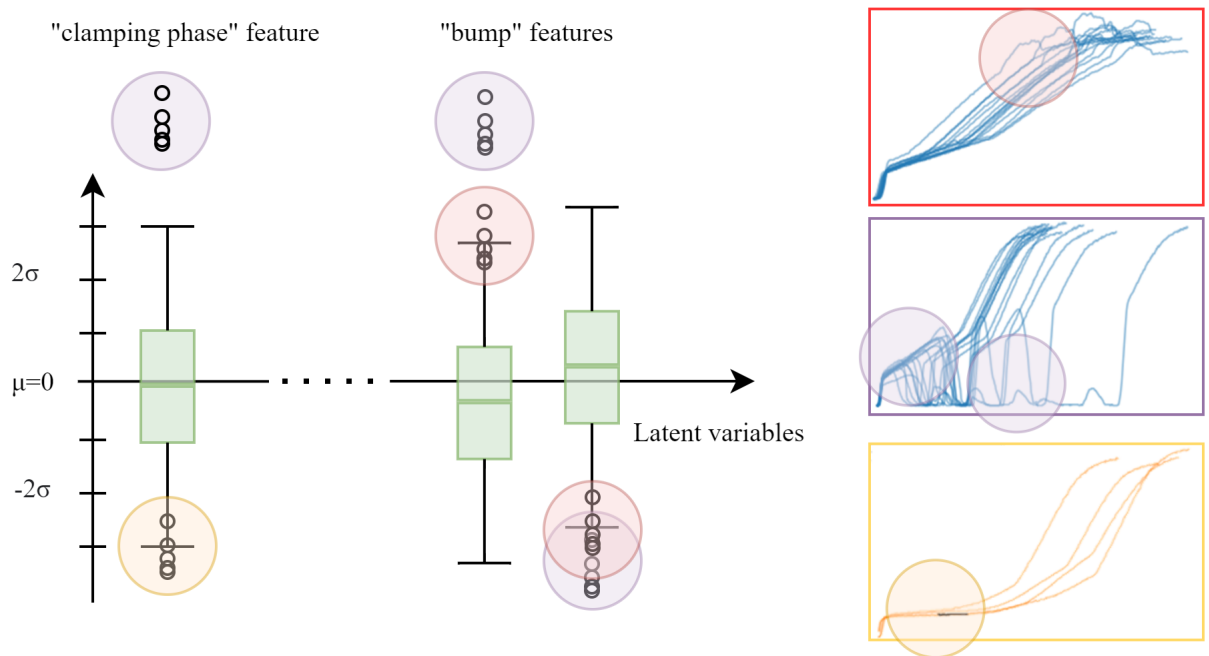


Figure 50: A (small) number of latent features correlate with visible features in the input data.

5.4 Continuous model retraining

5.4.1 Cost function for one-class classification

The semi-supervised approach discussed so far allows to identify anomalous objects in large unlabeled datasets. The need for this has been discussed in section 2. Based on this model, SMEs are asked to annotate objects that lie outside the thresholds of the latent variables. If an object outside the threshold T is labeled as normal, the model must change its learned representation of the majority class so that the point lies within this threshold. Both the reconstruction loss and the divergence loss discussed in the previous section are unsupervised loss terms i.e. they do not take into account the class a data point belongs to. Therefore, a third loss term is required to retrain the model based on user feedback. Intuitively, this loss should increase the further away the object is from the threshold, so that the gradient descent algorithm will adjust the model parameters Θ more if the point lies far away from what is considered normal. The simplest loss term $L_{integrate}$ that satisfies this requirement is a piece-wise function 7. Since the normal distribution has, by definition, a mean of zero, the loss is simply the absolute value of the feature vector x. Once the point lies within the threshold i.e. is classified as normal, the loss term is zero. Thus, uncritical anomalies will tend to cluster around the edges of the normal class (as they arguably should).

$$L_{integrate} = \begin{cases} |x| & if\,|x| > T, \\ 0 & otherwise \end{cases} \tag{7}$$

The distinction between critical and uncritical anomalies can be very subtle. By focusing only on retraining uncritical anomalies, we may unintentionally end up with a latent representation that incorporates critical anomalies into the normal class as well. To avoid this, it must be ensured that the model retains its ability to discriminate these anomalies from the normal class. Essentially, the model must not forget what it has previously learned. Therefore, a fourth loss term is defined using simple exponential decay function 8.

$$e^{-\lambda x} \tag{8}$$

The loss is the highest in the middle of the normal distribution and decreases as the object moves away. This ensures, that critical anomalies move away from the normal distribution and the loss never reaches negative values. The decay rate λ determines how far away these

object will be from the normal distribution once the model retraining is completed. Both loss functions are schematically shown in figure 51.

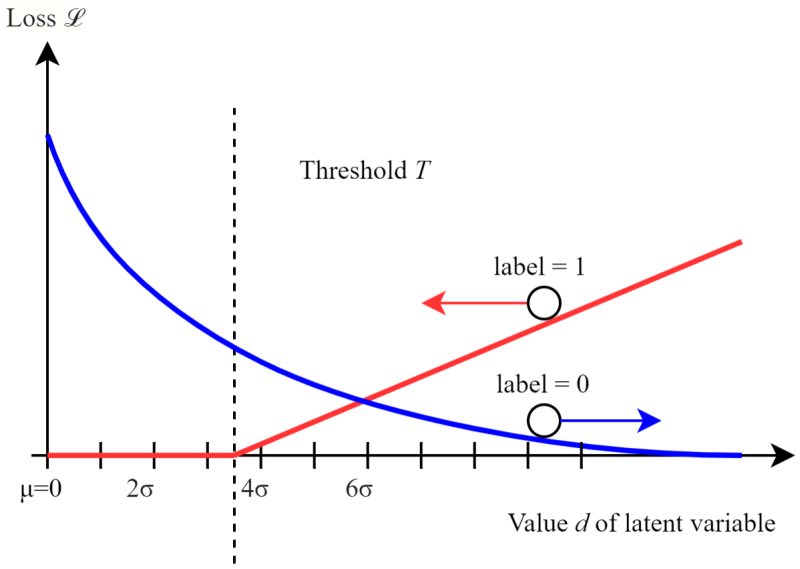


Figure 51: Cost function of the one-class LVM for anomaly detection.

Thus, the cost function of the one-class latent variable model contains four loss terms. Note, that both $L_{integrate}$ and $L_{discriminate}$ are *conditional* loss terms i.e. they are only applied to objects of their respective class.

$$L = L_{unsupervised} + L_{conditional} = L_{reconstruction} + L_{divergence} + L_{integrate} + L_{discriminate} \quad (9)$$

5.4.2 Retraining strategy

When retraining the model, its ability to recognize and analyze features should remain intact. Since the features in the input data have not changed, the aim of retraining the model should be limited to associating the right features with the normal class. Essentially, the model should adapt its decision boundary. However, the normal distribution of the latent variables must be maintained (for the previously discussed reasons). Therefore, model retraining is restricted to the layer that maps the output of the RNN to the latent variables using the conditional loss $L_{conditional}$. All other layers are frozen. This ensures, that objects of the majority class remain within the threshold, anomalies remain outside the threshold and the latent space remains normally distributed. The information flow of this retraining process is depicted in figure 47.

The retraining data is an aggregation of annotated data as well as unlabeled data of the majority class. The latter is included to ensure that the majority class remains normally distributed i.e. the model does not ”forget” what it has previously learned. During model retraining, the conditional loss is increased slowly to ensure a controlled and steady decrease in the training loss. If this is done too fast, the training loss will jump about about and the model may fail to converge. When done properly, the model will adapt its mapping to the latent space as shown in figure 52. It is important to keep a number of factors in mind when retraining the proposed model:

- Often, very little annotated data is available. Naturally, using a small validation set risks overfitting the model.
- The ability to retrain the one-class model for a very noisy majority class depends on the complexity of the mapping layer. Increasing the number of hidden states and the number of latent variables can increase the model’s capacity for retraining.

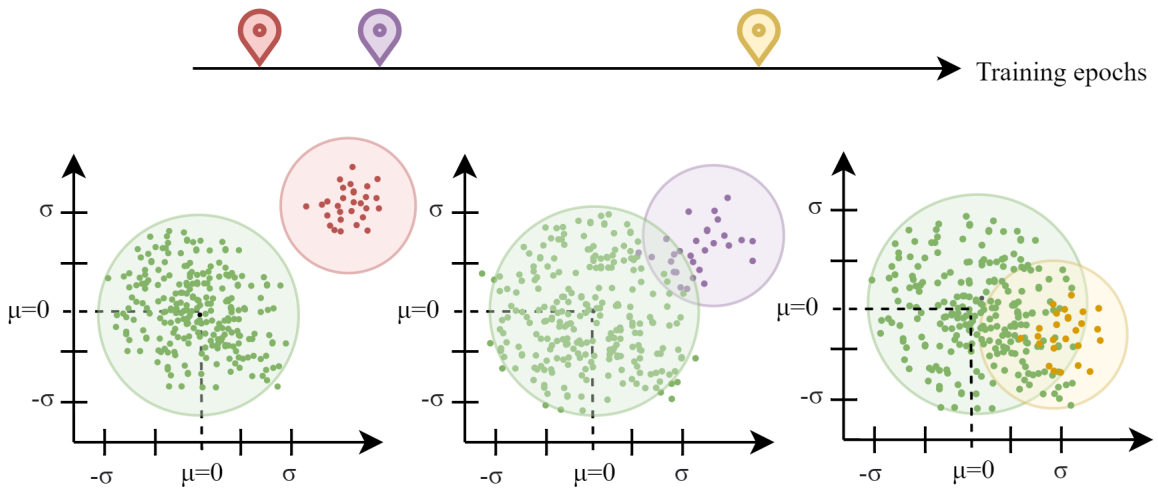


Figure 52: The change in the training loss and the latent space of the model during the retraining procedure.

6 Real-world system evaluation in the manufacturing domain

The primary focus of the work presented thus far has been the development of an approach to constructing one-class classification models for anomaly detection, whose (re-)training can

be effectively orchestrated to allow scaling it to a large number of processes. The previous two sections described two alternate approaches to do this. This section seeks to evaluate them on a system-level. Although the previous sections contained evaluations of their own, these were primarily concerned with motivating design considerations or benchmarking model performance in isolated experiments. To evaluate their *scalability* under real-world conditions requires a comprehensive field study. Thus, both approaches were implemented in parallel in a real-world manufacturing setting as part of an industrial ML project in the automobile industry that lasted for 20 weeks. The first subsection describes this setting and the process data. The second and third subsections summarize the workflow and software system used for (re-)training and managing the models. These were kept as generic as possible and not tailored to either approach to ensure comparability of both approaches using the same methods. The last section evaluates the ability to adapt the models and scale the system over the 20-week period.

6.1 Domain setting

The field testing was done for 72 different tightening processes in an engine assembly line of an automobile OEM. Tightening-related faults are the most common source of manufacturing-related recalls (itself the largest shared of overall recalls). The tightening processes were selected at random and accounted for approximately 15 percent of all tightening processes in the assembly line. Over the 20-week period, each process recorded $10^5 - 10^6$ time series measurements. The first subsection discusses the fundamentals of tightening operations that help the reader interpret the data. The second subsection discusses the data itself and explains the mechanisms behind a selected number of anomalous patterns detected during that period.

6.1.1 Fundamentals of tightening processes

Tightening of threaded fasteners is the dominant tightening technology in the automotive assembly industry. In most cases, a bolt is tightened into a threaded (blind) hole in the part to be assembled. The goal of the tightening operation is to induce a clamping force between the assembled parts by deformation of the bolt. Although this technology has been used in serial production for over a century, controlling the process in large-volume, low-tolerance environments remains challenging. The clamping force cannot be measured directly in a production environment. It is technically possible to measure the deformation of the bolt and calculate the tensile forces via the stress-strain relationship of the bolt material. However, this is extremely

challenging to do in production at scale. Methods exist to measure the elongation of the bolt via ultra-sound sensors [237] - however, this has thus far not been adopted to high-volume production. Instead, the process is controlled by measuring the tightening torque that correlates with the deformation of the bolt. Tightening a joint to a predefined torque in what is known as a torque-controlled tightening operation is the most commonly used method of process control. The challenge is to keep the conditions of the tightening joint constant so that this correlation is as reproducible as possible. Most of the torque energy is converted to heat due to friction, while only a small fraction is used to mechanically deform the bolt and the assembled parts. Approximately 50 percent of the energy is expended on friction at the interface between the bolt head and the bearing surface of the part and 40 percent on friction in the threads of the joint [238]. It is extremely difficult to control the various external factors that influence these frictional forces, such as manufacturing tolerances of the threads, amount of lubricant or sealant in the thread or the surface properties of the bearing surfaces etc. The amount of friction and deformation of the bolt and thus the final clamping force can vary considerably, often as much as 50 percent even in controlled environments [239]. To narrow this variance and ensure that a sufficient amount of torque energy is introduced into the joint, the bolt can be tightened past its yield point until it starts to plastically deform. This ensures, that a minimum clamping force is reached. The plastic deformation can be detected by monitoring the tightening torque: the torque curve will level off even as the angle of rotation is increased. This is because the bolt material will begin to flow and relax the built-up stress in the bolt. This is called an angle-controlled tightening processes and is predominantly used in the automobile industry for safety-critical tightening connections [238, 240]. Figure 53 shows the progression of the tightening torque over the angle of rotation of the bolt for typical tightening processes in an automotive assembly setting.

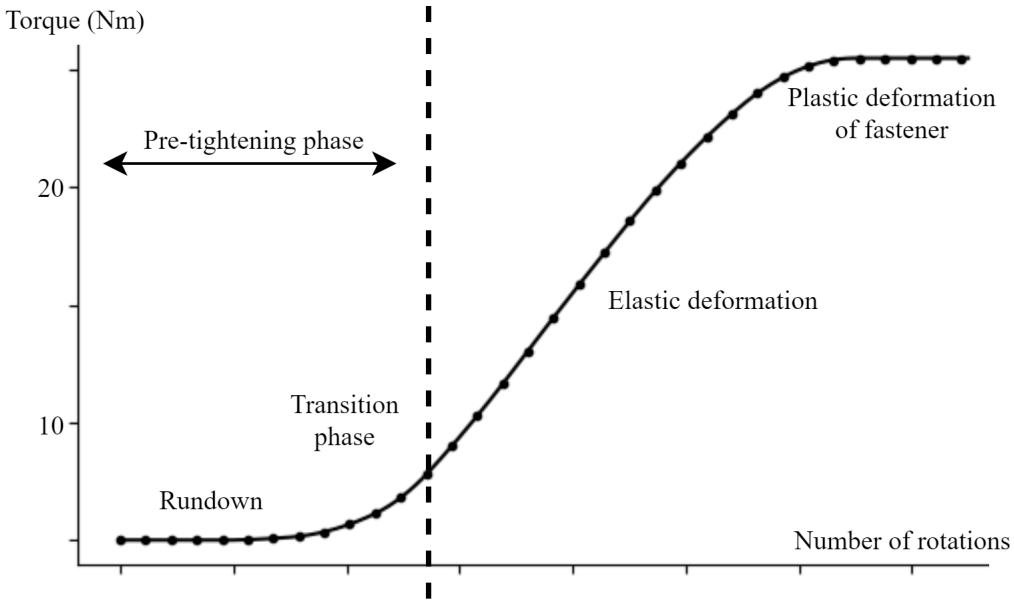


Figure 53: Schematic torque-angle signature of typical tightening processes.

The progression of the torque over the angle of rotation is referred to as the torque-angle signature of a joint. The area under the curve is the energy introduced into the joint that is dissipated as frictional heat or used to mechanically deform the fastener and the tightened parts [240]. The process is usually split into a pre-tightening operation and a main torque- or angle-controlled tightening operation and can be divided into four phases. Pre-tightening includes the rundown phase before the fastener head contacts the surface and the subsequent alignment phase, during which the surfaces of the parts are drawn together and aligned. The next phase is the elastic clamping range during which there is a nearly linear relationship between torque and angle. Torque-controlled tightening processes are terminated once the torque reaches a predefined value within this range. For angle-controlled processes, this is followed by a post-yield phase during which the fastener plastically deforms. The inflection point is characterized by the underhead embankment of the fastener [240].

Monitoring the angle-torque signature of a joint is the predominant way of process control in high-volume manufacturing settings. The process data is often readily available, as it is often the primary (and only) means of process experts to ”look inside” the process. In the case of safety-critical joints, there are also legal requirements to record and store this data. Thus, the availability of tightening process data in the automotive industry is essentially guaranteed. Since most vehicle recalls (by number) are attributable to manufacturing defects, and tightening faults account for most of those defects, the analysis of torque-angle signatures as a means

of preventive fault detection has merit beyond this particular setting.

The tightening processes in the assembly line that were monitored during the pilot phase are completely automated. The work-piece carrier (WPC) moves along the assembly line on a friction roller conveyor (FRC). After the WPC reaches the operations position within an assembly station, a stopper fixes its position on the FRC. The bolt is either fixed to the work-piece manually in a previous operation or fed from the separation unit through a flexible tube to a holding mount in front of the tightening spindle via compressed air. The tightening spindle is lowered at low rotating speeds during the finding phase. Once the nut slips into the hex socket and/or the bolt into the threaded hole of the work-piece, the actual tightening process described in the previous section begins. As discussed, the torque is measured by the machine controller to control the tightening process. After the process is completed, the machine controller passes the data to the digital controller of the assembly station, that stores it in an internal buffer storage, from which it is transferred to the centralized control system (CCS). The buffer storage ensures that data can be stored temporarily and resent upon request from the CCS so that it is not lost if network bandwidth limit is reached.

6.1.2 Tightening process data

Over the course of the 20-week evaluation project over 68.000.000 time series were recorded by the 72 processes and analyzed by the ML system. Fewer than 1 in 10.000 objects were annotated by SMEs during the pilot phase (less than 0.01 percent), approximately 100 objects per day on average. For the experiments described in section 6.4, all data that is not annotated is assumed to belong to the majority normal class. Both SMEs and data scientists spent considerable time searching through the data set for objects similar to the identified anomalies. It is unlikely, therefore, that a large number of objects of interest were omitted.

To get a sense of the variability between different tightening processes and the variance within the same process, representative normal data from the three process types discussed in the previous section is plotted in figure 54.

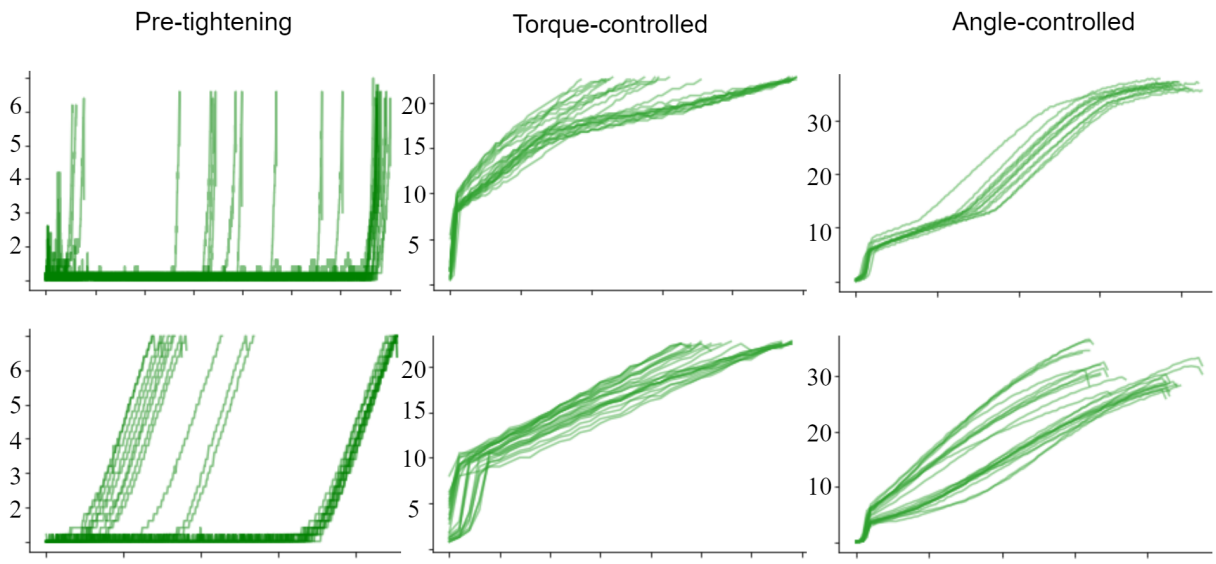


Figure 54: Process data of pre-tightening and angle- or torque-controlled tightening operations can be very different.

The first columns shows two randomly selected pre-tightening processes. Both the shape and length of the patterns vary significantly (the latter by the order of one magnitude). The wave-pattern visible during the rundown phase is due to cyclical changes in the friction between the thread and the bolt. It is visible exclusively for the pre-tightening processes as the fastener is not yet under tension. This is a normal pattern in the torque signature of pre-tightening processes - for other processes a drop in the otherwise monotonically rising torque signatures would constitute an anomaly. The second and third columns show torque and angle-controlled tightening processes, respectively. Each process exhibits a characteristic shape and variation, depending on the process and joined parts. For example, the angle-controlled process in the first row exhibits a characteristic bend between two elastic deformation phases. The first phase corresponds to the deformation of the (stiffer) joined parts while the second corresponds to the deformation of the fastener itself. A similar phenomenon is visible for some torque-controlled signatures in the first row. The fact, that this is not the case for all instances suggest that there was a change in the material properties of the assembled parts or the fastener (or both) during the time period from which this data is taken.

Over the evaluation period five anomaly types were identified that were confirmed to corresponded to quality faults. Figure 55 shows three of these anomalies (red). In the first subplot (blue), the tool slipped off the bolt during the tightening operation due to misalignment between the tool and the fastener (cf. section 4.4.2). Although obviously different from normal

process operation, this is not an interesting anomaly from an SME perspective. In comparison, the three subplots on the bottom (red) correspond to different anomaly classes detected in the same and similar tightening process. In the first subplot, the fastener is retightened after an initially failed attempt. This is problematic for an angle-controlled process, since the fastener plastically deforms during the tightening operation and must not be reused. The second subplot shows the presence of a foreign object in the thread that causes an anomalous ”bump” in the torque curve. In the third subplot, a pinched gasket prolongs the compression phase of the joined parts and delays the onset of the elastic elongation of the fastener. This means that the tightening process is terminated by the process controller before the the bolt is sufficiently elongated, resulting in an insufficient clamping force, leakage from the damaged seal of the gasket and the build-up of mechanical stresses in the assembled part as it is bent over the bulging gasket. All this can be gleaned by SMEs from the lack of the characteristic transition from the elastic compression of the joined parts to the elongation of the (stiffer) fastener, that causes a change in the slope of the torque signature. A somewhat similar pattern is visible when too much sealant is added to the thread of the fastener, resulting in increased friction in the threaded connection. This results in a prolonged pre-tightening phase that ”stretches out” the curve. Although similar (and anomalous), this mechanism is not interesting and was added to the training data for the respective one-class model. The figures and discussion aimed to showcase the large variance of the tightening process data and the need to incorporate domain knowledge when training these models.

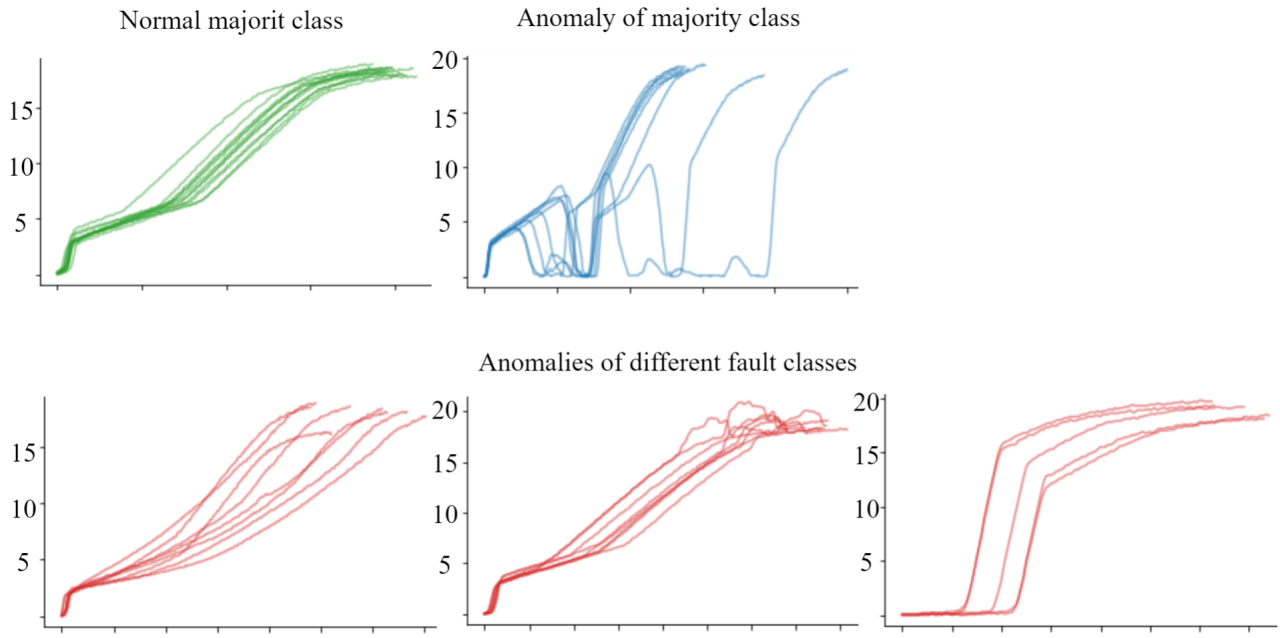


Figure 55: Characteristic objects of the majority class and anomaly class of a single tightening process.

6.2 Machine learning operations workflow

This section provides an overview of the workflow that the data scientists followed during the evaluation phase. It describes how the modeling approaches developed in sections 4 and 5 were applied to train and deploy ML models into production. While these tasks typically fall in the responsibility of a data scientist, the boundaries between the work of other parties involved in the ML Ops process e.g. analysts and data and software engineers may vary significantly. The ML Ops workflow can be broadly subdivided into three stages: algorithm development, model training and model serving. This is schematically depicted in figure 56.

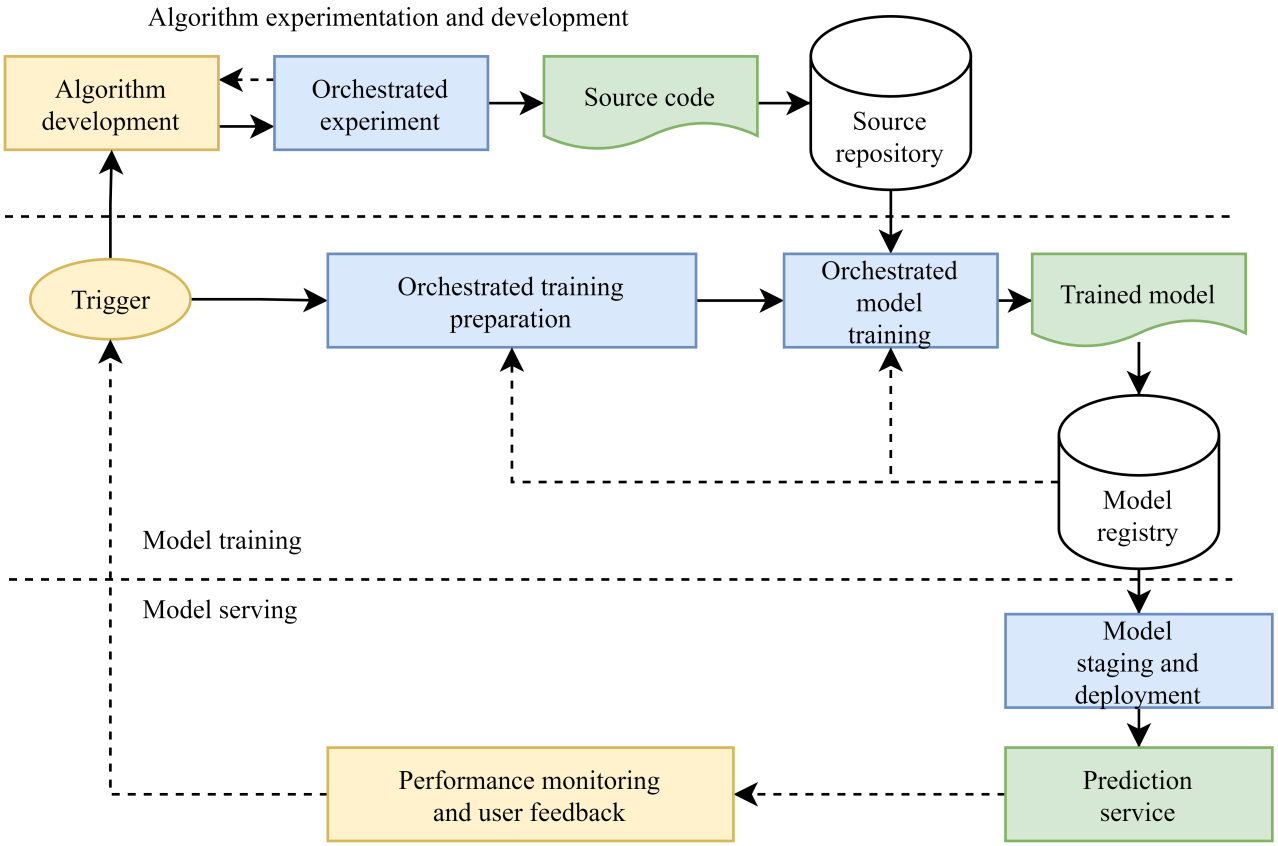


Figure 56: Schematic workflow for ML model life cycle management.

During the exploration phase, data analysts and data scientists define the problem and explore the data. Once a suitable ML problem is formulated, an experimental phase follows in which ML models are trained and tested using off-the-shelf or purpose-built algorithms (such as the ones developed as part of this work). As discussed, algorithm selection is typically based on a combination of personal preference of the data scientist and model performance. To make this iterative process more efficient, an orchestrated experimentation pipeline is set up to test the results of new development iterations on predefined test sets. Once a suitable algorithm has been found it is registered in the source code repository. Models are then trained using this source code and stored in a model registry together with information such as the version of the source code used, preliminary model performance and the data used for training the model. A separate deployment and serving service scans the registry and deploys the model as a stand-alone prediction service. Once in production, SMEs provide feedback on the prediction services by annotating the detected anomalies. This feedback is the central performance indicator used by data scientists to trigger debugging or retraining of the models. If satisfactory performance cannot be achieved, the experimentation and development stage is reiterated. Essentially, the three stages of the MLOps process are connected via two continuous feedback loops.

The focus of this section is to explain the tasks of training models and maintaining them in production in order to provider the reader with a sound understanding of how the evaluation phase was conducted. Discussion of the algorithm development process is omitted. It is the stated objective of this thesis to provide an approach to model training that reduces the iterative nature of this process - the approaches laid out in the previous sections 4 and 5 are the *result* of this process.

System monitoring The goal of performance monitoring is to detect and remedy model degradation as soon as possible. This is vital to ensure user's confidence in the system. Degraded models cause excessive labeling effort for users and may, in extreme cases, render the prediction service entirely unusable. If this happens frequently, users will lose confidence in the system. Thus, ML practitioners must try to detect model degradation before it reaches a degree that users will consider unfeasible. Model degradation is caused not by changes to the model but by changes in the data:

- A productive model classifies objects as anomalous that are labeled as uninteresting by the user i.e. the model prediction is different from what the user expects. As long as the model is not retrained, it will continue to detect these anomalies.
- The statistical properties of the data may change (concept drift). This can result in the one-class model detecting a larger-than-normal number of (uninteresting) anomalies. This change may occur suddenly or gradually over time.

In both cases, the model needs to be retrained. Judging whether a model is degraded in either of these scenarios requires the definition of thresholds on the number/share/frequency of (uninteresting) anomalies detected in a given time frame. In practice, the choice of threshold largely depends on the capacity of SME to label the data and of data scientists to retrain degraded models. In practice, it can be challenging to keep track of the performance of a large number of models. During the evaluation phase, data scientists interacted with the ML system via a web front-end of a purpose-built application. The application is structured around the model life cycle management workflow depicted in figure 56. The landing page of the application gives the data scientist a quick overview of the state of health of the system and serves as the starting point of the workflow. It is depicted in figure 57.

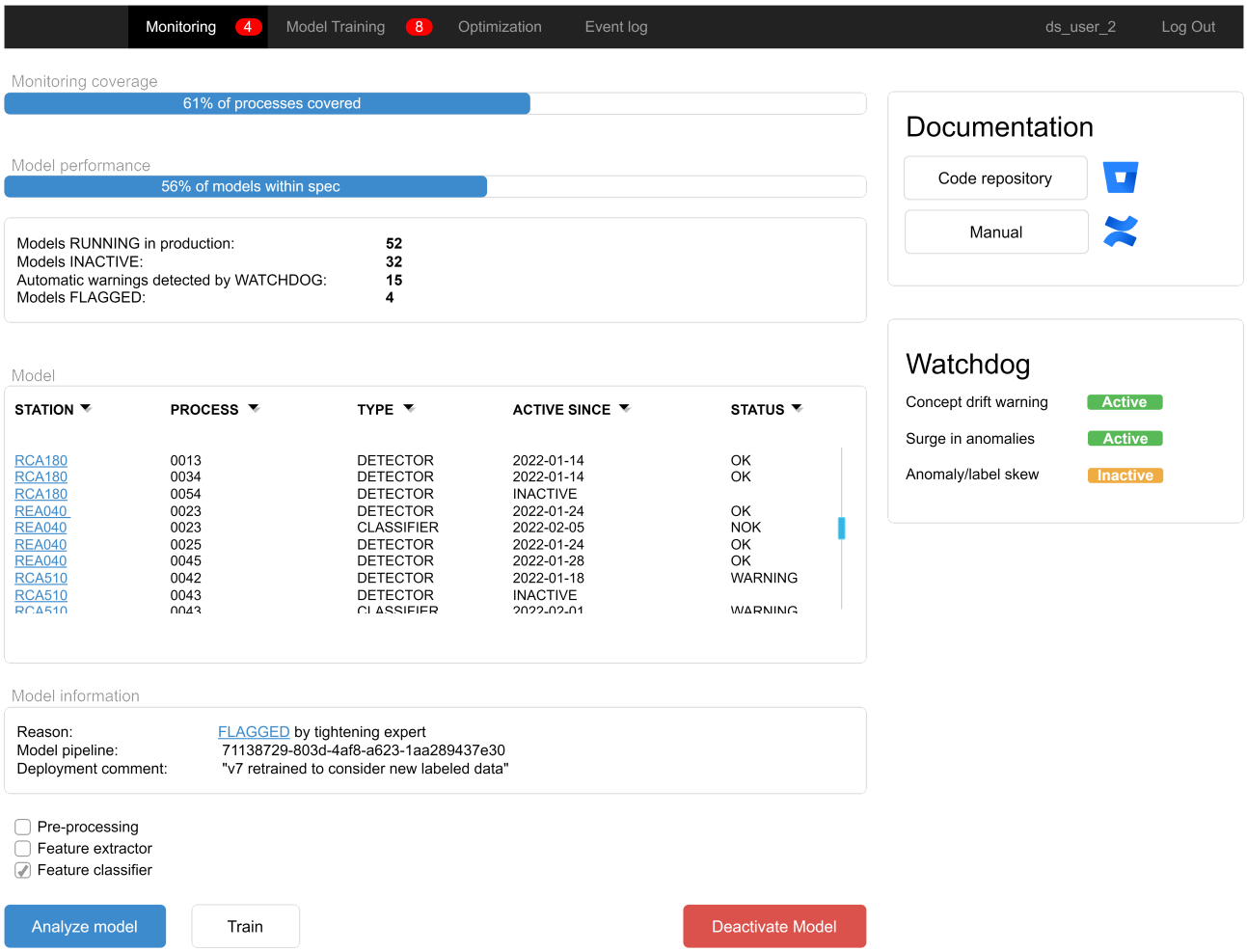


Figure 57: Monitoring page of the web front-end of the purpose-built ML Ops application.

The progress and performance of the system is summarized at the top of the page. The monitoring coverage summarizes the share of processes that are monitored by one-class models or specific classifiers (or both). The model performance summarizes how these models fare in production, broken down into (1) models that exhibit nominal performance within (user-defined) specification limits, (2) models for which an automated warning indicator was detected (a so-called ”watch dog” that monitors user-defined thresholds on the number/share/frequency of uninteresting anomalies) and (3) models that have been manually flagged by SMEs as problematic. This gives the data scientist an overview of the progress related to scaling and adapting the models. Below this summary is a more detailed table that gives a high-level status of the state of health of each registered model. The status can be one of three categories: ”OK”, ”Warning” or ”NOK”. There are three separate feedback loops that can change this status:

- If SMEs recognize a problem with a particular prediction service, they can manually flag the model through the same (separate) SME web front-end that is used for annotating detected anomalies.

- If the number of anomalies exceeds a threshold, a concept drift warning is raised. While plenty of research on concept drift detection exists, these approaches are usually based on (more or less arbitrary) thresholds applied to statistics derived from the data. Rather than monitoring these statistics, the system applies a threshold directly to the number of detected anomalies.
- If the number of anomalies labeled as uninteresting make up a significant share of the detected anomalies (anomaly/label skew).

Model training and serving Training ML models is a central task of ML system operation. To make this process feasible, the degree of automation of the training workflow must be very high. The frequency of training new models (when scaling the system), retraining existing models (when adapting the system) and deploying these models into production is much higher during the operating phase than during the experimental development phase that precedes it. Once the system reaches a certain scale, these processes must be automated for the workload to remain tractable (cf. the definition of scalability in 2). Each process contains a series of tasks that are automated to varying degrees over the course of development. Automating the process requires automating this series of tasks including all incoming and outgoing triggers, dependencies and configurations. This is known as *orchestration*.

Definition 14 *Orchestration is the automated configuration, management, and coordination of computer systems, applications, and services [241].*

Consider the orchestrated training training pipeline depicted in figure 58. When the pipeline is triggered, the first task is to validate the data. A connection to the data source (in this case a structured database) is established and the data checked for any inconsistencies that do not conform to the expected data format such as missing or undefined values. The validated data is then split into training, test and validation set according to a predefined split criterion. Next, the source code of the algorithm is pulled from the source code repository and the model training process initiated. If model performance is satisfactory, the model and its metadata is pushed to a model registry. Otherwise, the model hyperparameters are adjusted according to a predefined training strategy (e.g. a greedy search or a brute-force grid search) and the training process rerun until a performance threshold is reached. If no suitable parameters are found, the pipeline is terminated and an issue raised to the data scientist. This process reduces

the required manual input by implementing a predefined (and reproducible) training strategy using algorithms from the source repository.

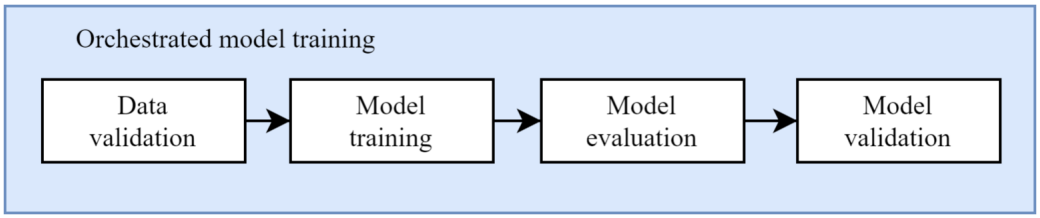


Figure 58: Automated tasks in an orchestrated training pipeline.

ML Ops poses much more stringent requirements on version control than regular software engineering. Conventional version control systems are centered around versioning of the source code. In addition, version control for ML systems includes versioning of the trained model pipeline to allow rollback to a previous version, the configuration and metadata that was used to train this model (this includes the version of the ML algorithm and model hyperparameters) and the data used to train and validate the model. The model registry serves as a central repository that stores the ML model along with training artifacts like metadata about the training job as well as training/test/validation data. Tracking this information is vital to be able to reproduce a specific model pipeline at a given point in time. This, in turn, is necessary to evaluate changes to the models and algorithms and allow efficient root cause analysis and debugging. The model registry combines the functions of a conventional version control systems and an artifact repositories for traditional software systems. Once the model is pushed to the registry, it is ready for integration and deployment as a prediction service. The process of model integration and deployment is completely automated and triggered as soon as a new model is pushed to the registry (known as continuous model integration).

Optimizing the model landscape An important task for data scientists is to optimize the model landscape. The aim is to find a trade-off between lowering the complexity of both the individual models and the overall system. The goal is to simplify system operation by making it easier to retrain models and leveraging labeled data from one process for training models for other processes. Data scientists routinely check whether models monitoring different processes can be unified into a single model (vertical scaling). By analyzing the (annotated) process data and gathering feedback from SMEs, data scientists may identify processes that are very similar to each other. This can be verified by cross-validating the performance of these models with data from different processes. The front-end application contains a module for this task,

allowing the data scientist to test the performance of registered models in the model registry for a selected process. If a model is able to classify objects with acceptable accuracy for multiple processes, the data scientist can continue with one model and retire the rest. This routine task of model consolidation is part of the standard workflow in figure 59. This same procedure can also be applied in cases where annotated data of a new process is available from the start. Instead of training a new model, data scientists can check if a model exists in the model registry that delivers adequate performance. If this is the case, the model can be directly applied to the new process. Essentially, both of these processes are a form of transfer learning that avoids "learning the same thing twice".

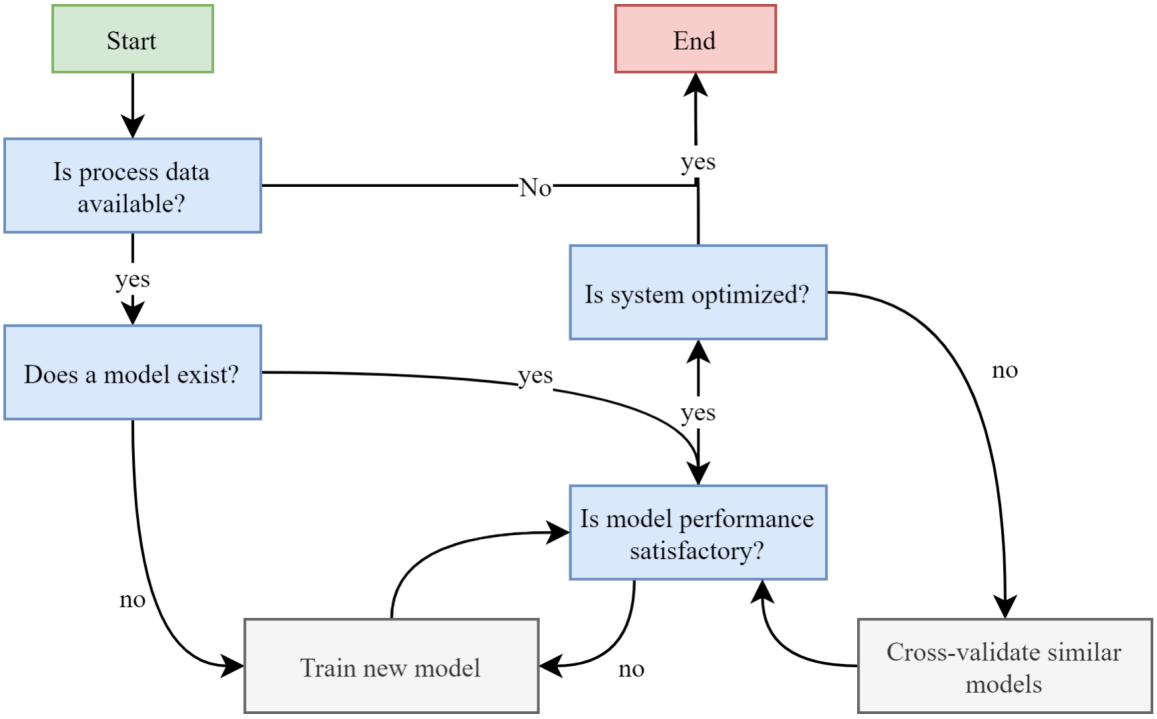


Figure 59: Decision diagram of the model training process

While fewer models makes the monitoring task easier, it tends to make the retraining task more difficult. While training a new model for a single process is relatively straight-forward, retraining or replacing an existing model comes with data dependencies. It is important to ensure that changes to the model do not adversely impact the classification performance for past data. This data dependency increases with the number of processes. In some cases, the process of model retraining and debugging is more tedious than maintaining separate models for each process. In such cases, data scientists will branch a new model from an existing model. As discussed in section 2, whether it makes sense to use a single model for multiple processes or vice versa depends on the situation.

6.3 System architecture

This section discusses the architecture of the ML Ops software system that was used to train, deploy and manage the ML model pipelines. The system is a technical (software) implementation of the workflow described in the previous section. The software was developed specifically for the evaluation phase of this research. The source code and documentation of the database schema and service endpoints are open-sourced [167]. Without this system, a real-world evaluation of the scalability of the proposed algorithms would not have been possible. The system is completely independent of the implemented ML algorithm. Thus, any difference in system scalability and operability should be attributable to the proposed ML algorithms.

6.3.1 Graph-based model management

To effectively monitor the large number of different processes in a production system, a large number of model pipelines is required. This number should generally be as small as possible to reduce the complexity of the system. For processes that are sufficiently similar, it should be possible to easily reuse (partial) pipelines as much as possible. On the other hand, keeping an overview of these pipelines and their mutual dependencies quickly becomes difficult for large-scale ML systems. Without a systematic process, model retraining quickly turns into a time-consuming process of retracing previous steps and manually assembling training data sets. To avoid this, the software uses a graph-based approach to model management that allows to keep track of and visualize these dependencies.

Raw data recorded for each production process is processed by a single model *pipeline*. A model pipeline is a sequence of models, that process the data from raw data input to prediction output. Typically, this pipelines consists of data pre-processing, feature extraction and classification. If we consider the models of a pipeline as nodes and the data flow between these models as edges, then the pipeline can be represented as a directed (acyclical) graph. Each node processes the data and passes it onto the next downstream node. This same concept can be extended to the history of each node (model) in the graph. This is schematically shown in figure 60.

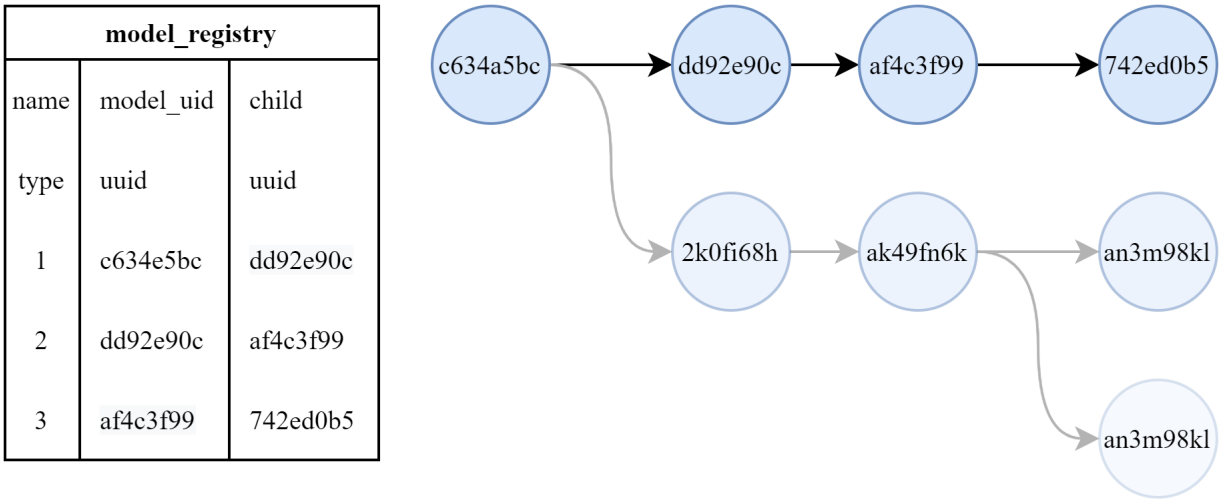


Figure 60: A model pipeline (including its history) can be interpreted as a simple graph.

Since the real-world process outputs raw data that is the input to the pre-processing node, the process can be considered an upstream node that connects to the first node in the pipeline graph. Thus, the root node of the graph generates the input data and the leaf node of the graph outputs a prediction. A root node can be connected to multiple pipeline nodes so that the same input data is processed by multiple classification pipelines. Additionally, multiple nodes from different pipelines can share the same downstream node, irrespective of their root node. In fact, *any* node A can be connected to any other node B in the graph, irrespective of the root node so long as (1) node B is a downstream node (e.g. a feature extraction model must be followed by a classification model) and (2) the connection does not create a loop in the graph i.e. two nodes pointing at the same node B must stem from different root nodes. This makes it possible to intuitively depict the dependencies between models across different processes. This is schematically shown in figure 61.

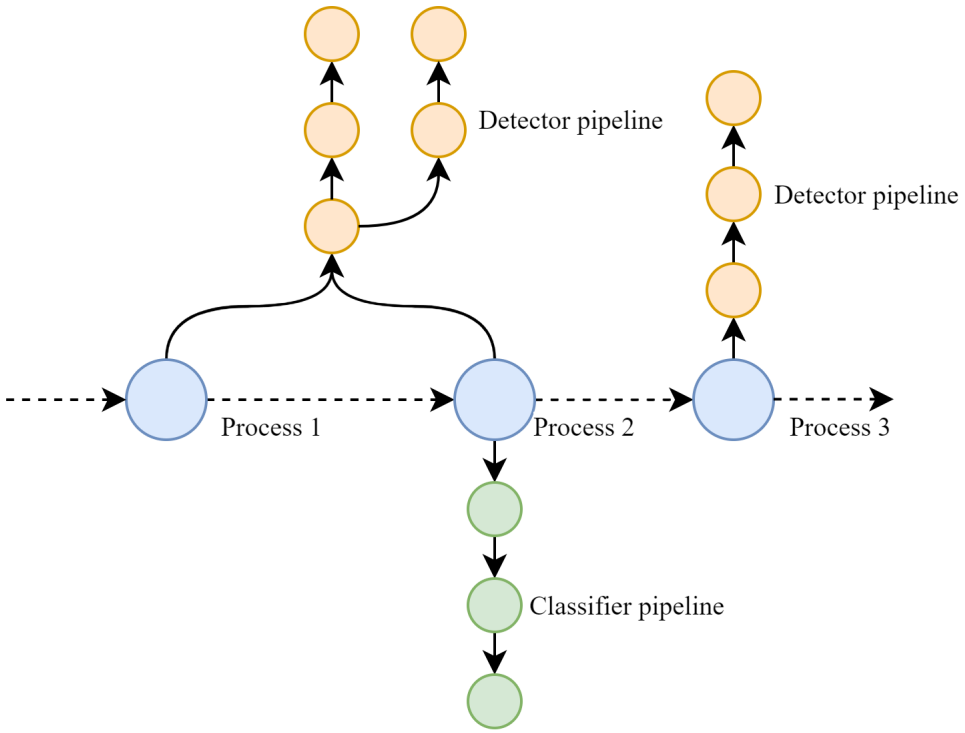


Figure 61: The same model pipelines can be used to monitor multiple processes.

For example, a process may exhibit two very different fault types that require separate models for feature extraction and classification. However, the pre-processing of the data may well be similar, which means both pipelines can branch from a shared pre-processing node. For sufficiently similar processes, the corresponding root nodes may be pointed directly at a shared pipeline. This graph-based approach to model management has many desirable characteristics:

- Making changes to a model pipeline is as simple as updating the pointers between the nodes in the model registry. When a productive model is replaced by a new version, the old pointers to and from the model are deactivated and new pointers are added. This makes it possible to reconstruct the pipeline that was active at any given time, allowing to roll-back changes in the pipelines if performance degrades.

- For classifiers that are used for multiple processes, all relevant training data can be found by simply retracing the graph to find all relevant root nodes, starting from the leaf node in question. This is schematically depicted in figure 61.

- The model graph is an intuitive visualization of the history and dependency of the model pipelines that make it easier to keep an overview of the system.

6.3.2 Model and data services

The ML Ops software system is made up of three separate services:

- The Model Build Service (MBS) orchestrates the model training pipeline and ensures that active models registered in the model registry are built and deployed into production.
- The Model Management Service (MMS) administers the model graph and ensures consistency of the dependencies.
- The data provisioning service manages the data warehouse that stores the production data, the prediction results of the ML models and the feedback provided by the SME.

These services communicate with each other using standard protocols and application programming interfaces (APIs). APIs allow two independent services, such as the model build and management services, to communicate with each other by exposing their functionality via predefined endpoints. The reason for this service-oriented architecture is twofold: restrict data-heavy and compute-intensive operations to the back end to speed up system interaction and ensure, that parts of the system can be taken ofline/extended while keeping the rest up and running. This is especially important for the prediction services of the productive models and the data provisioning service. Unavailability of the system, an unresponsive front-end or a loss of data quickly erodes confidence of SMEs in the software. The system architecture, including its services, databases and data flows are depicted in figure 62.

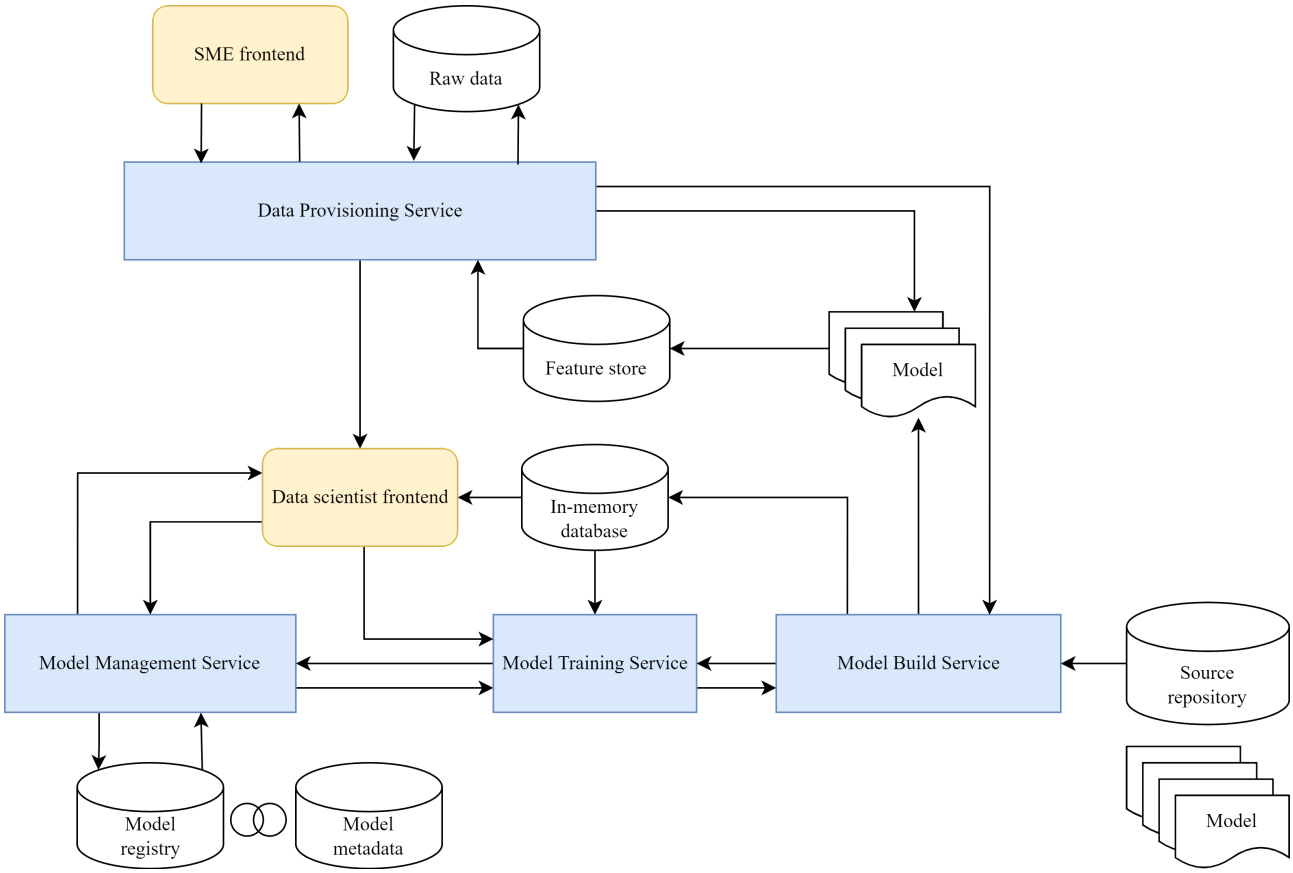


Figure 62: The data flow between the services, front-ends and databases that comprise the ML Ops software system.

To explain how this system works, we will consider three training scenarios: (1) unsupervised training of an initial one-class model (2) semi-supervised retraining of a one-class model/classification model and (3) simplification of the model landscape to improve operability.

- The ML practitioner sends a training request to the MBS via the front-end application. The service prepares a detailed training request that specifies the train and validation data to be used for model training, hyperparameters for data pre-processing and the type and version of the ML algorithm. The request is then passed to the build server, which executes the training pipelines. It pulls the source code of the algorithm from the code repository, parameterizes the algorithm, retrieves the training data from the data service and starts the training job in a dedicated container. This enables the build server to run multiple training jobs in parallel. Once the training job is completed, the build server stores the trained model and the model metadata in an in-memory database that is accessed by the front-end application. While the data in all other databases is never deleted (only deactivated), the intermediate training results in the in-memory database are deleted at the end of the training session. The ML practitioner may inspect the

results and opt to rerun the training cycle or approve the model. The approval request is sent to the MBS that retrieves the model and its metadata and sends it to the model management service which registers the model in the model registry. The activation of a new model in the registry triggers the execution of a deployment pipeline. The model is served in a separate container along with a micro-service that pulls new data from the data warehouse through the data service. The extracted features and final results are stored in the feature store and the latter passed onto the data service to be stored in the data warehouse. The SMEs are then able to analyze and label the detected anomalies through a dedicated frontend application. Information about the number of detected anomalies and the discrepancy between predicted labels and SME feedback is continuously monitored by data scientists.

- The ML practitioner may wish to retrain a model pipeline due to low or degrading performance. The retraining process will generally start with the leaf node and move upstream only if sufficient improvement is not possible. This way, changes to the system are as big as necessary and as small as possible. This becomes more important for vertically scaled models that have many data dependencies. The MBS pulls all upstream models from the model management service and specifies the combined test, train and validation data from all root nodes. The build server handles the training request, reconstructing the upstream pipeline to train the downstream pipeline. The approved model pipeline is registered in the registry and the old pipeline is deactivated. The process of building, deploying and serving the model is analogous.

- To simplify the model landscape, a single model (pipeline) can be trained for multiple root nodes. Suitable pipelines and root nodes are identified based on recommendations made by SMEs or the system itself. The system can recommend to combine existing pipelines by evaluating the hypothetical performance of a pipeline for another root node. The micro-services of the productive pipelines are extended to get additional data from other processes. This is a brute-force approach of combining existing pipelines with (other) existing processes. The results are saved in the feature store and can be analyzed by the data scientist. If the performance of pipeline A for process B is similar that the performance of pipeline B, chances are that a combined pipeline will suffice. Due to the large amount of data traffic, this comparison is not run continuously but triggered manually by the data scientists.

6.4 Evaluation

6.4.1 Model adaptability

A vital part of ML system operation is the ability of data scientists to adapt the one-class models in production using the two approaches proposed in section 4 and 5. Model adaptability is determined by the ease with which data scientists are able to retrain models using annotated data provided by SMEs. One way to evaluate and compare the adaptability of the approaches would be to deploy two model pipelines at the same time (one for each approach), have SMEs label the predictions of both models and evaluate model performance after retraining. However, this was deemed unfeasible during the evaluation phase, as data scientists continuously changed the model landscape. This meant that some models were scaled to a much higher degree than others, drawing on more training data and/or covering more processes. Comparing the adaptability of the approaches based on differently scaled models would essentially be an apples-vs-oranges comparison. Instead, the adaptability of both approaches was evaluated via a controlled experiments using a shared data set as follows:

- The data set described in section 6.1.2 is split into 10 subsets, each containing *one* anomaly class and all data of the majority class (of the respective process). Five of the anomaly classes were critical, the other five were not. This yields a one-vs-all data set for a binary classification problem for each anomaly class.

- Both a DL and DM one-class model is initialized using a randomly sampled subset for each data set.

- The data set was analyzed by each model and the 100 most anomalous objects added to the training data for the next iteration, together with their (known) labels. This number corresponds to the average number of objects SMEs labeled in one session and is larger than the number of anomalies in each data set (i.e. a precision of 100 percent is possible on the first iteration). Objects of the normal majority class and uninteresting anomalies were labeled as 0 and critical anomalies were labeled as 1.

- The model is retrained using the labeled training data. At each iteration, the performance of the model is recorded and the results labeled and added to the training data. The retraining process is repeated for 5 iterations, after which no more improvement could be detected.

Both approaches implemented the training strategy for the unsupervised/semi-supervised model initialization and the supervised model adaptation that proved the most effective during the evaluation phase. These strategies included automatic hyperparameter tuning and (in the supervised case) cross-validation that required no manual input. Since the manual effort is the same (i.e. none) for both approaches, the evaluation of the adaptability of both approaches can be reduced to their respective performance. A discussion of the effort recorded during the evaluation for each approach is included in the next section.

The experiments were repeated ten times for each approach and data set in order to mitigate the influence of random model initialization and data sampling. Figure 63 shows the consolidated results for the DM and DL approach for all 10 anomaly classes, where the average and range of the model performance is recorded for each training cycle. For critical anomalies, the precision of the model is calculated over the retraining cycles, as the models learned to discriminate an increasing number of objects from the target class. The precision (of a detection model) is defined as the fraction of relevant instances among all detected instances (in this case the 100 most anomalous objects). For the uninteresting anomalies the false positive rate (FPR) was calculated. The FPR is the fraction of anomalous objects of that class among the detected instances. The advantage of this calculation is that the starting performance for the model is the same for both anomaly types, allowing a more intuitive comparison of the two approaches.

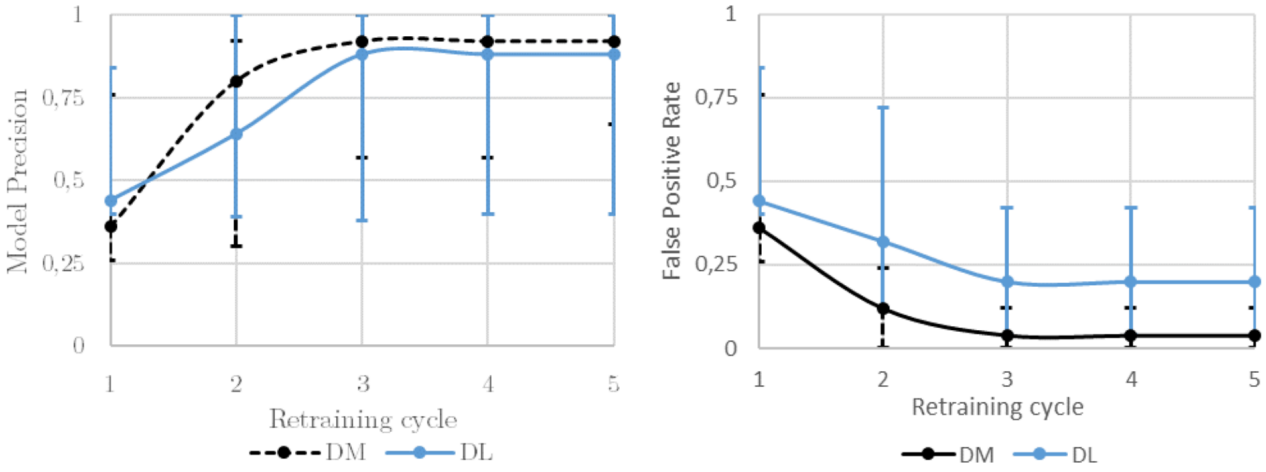


Figure 63: The precision of one-class models retrained to discriminate critical anomalies (left) and the false positive rate of the same models retrained to include uninteresting anomalies in the normal class (right).

Both the DL and DM approaches can be used for unsupervised training and supervised adaptation of one-class models based on incrementally provided feedback. This is not surprising,

given the results of the benchmark studies and field testing in the previous sections. The performance of both approaches is sufficient to be practically feasible. Starting with a completely ”fresh” unsupervised model, every third objects that an SME could visually inspect in a single session belongs to a characteristic anomaly class. While these are welcome results, there are a number of factors in a productive setting that reduce model effectiveness:

- This experiment considers a data subset consisting of a one-vs-all classification problem.
- Productive models that are scaled across multiple processes come with more complex data dependencies that make retraining more difficult.
- Based on the results, DL models are *relatively* better at discriminating a critical anomaly from the normal class than adapting the normal class to incorporate uninteresting anomalies. In fact, this has been observed consistently throughout the evaluation phase and becomes *more* pronounced the more (different) uninteresting anomalies are added to the normal class.

Thus, the performance depicted in figure 63 is representative of an ideal scenario that likely overstates what an ML practitioner will encounter in a productive setting.

6.4.2 System scalability

So far, the adaptability of the models were evaluated in isolated settings by looking at model performance. In this section, the scalability of the proposed approaches is evaluated on a system level based on data collected during the evaluation phase. As discussed in section 2 the scalability of a system fundamentally depends on the ability to extend system capacity using available resources. In the case of an ML system for fault detection of the kind discussed in this thesis, the constrained resources are the data scientists required for system maintenance and SMEs for data annotation. Therefore, evaluating the scalability of the proposed approaches requires a large-scale system-level evaluation.

Over the 20-week evaluation period, the models were initialized, adapted, debugged and/or scaled once during fixed development cycles lasting one week. An average of 25 hours were expended during each development cycle. All development was done by a single data scientist that was not involved in the development of the algorithm or the ML Ops software system and had no prior experience with tightening processes. The methods and functions of both

approaches were integrated into the same abstract class, allowing the data scientist to use the same methods and procedures for model training. All of these external variables were controlled to ensure that the results were not subject to personal bias of the data scientist. Additionally, it allowed to field-test the developed model approaches and ML Ops software system and judge its generalizability to generic settings.

One important consideration to evaluate the scale of the system is to look at the number of active models in production. This is an indicator of both the amount of time spent by data scientists on system operation as well as the amount of time spent by SMEs on data annotation. A large number of separate models tend to increase the need to annotate more data, as they need to learn similar representations instead of ”sharing knowledge”. The number of processes was increased in three phases from initially 40 to 60 to finally 72, as shown in figure 64. The total number of productive models at the end of each development cycle is plotted for both approaches. At the start of the pilot phase, the data scientist deployed a separate model for each process (this is the default approach to model training if no annotated data is available). As annotated data became available, the data scientist was free to branch or merge different models. To quantify this activity, the ratio of the number of processes over the number of productive models (i.e. the average number of processes monitored by a single model) is depicted in figure 64.

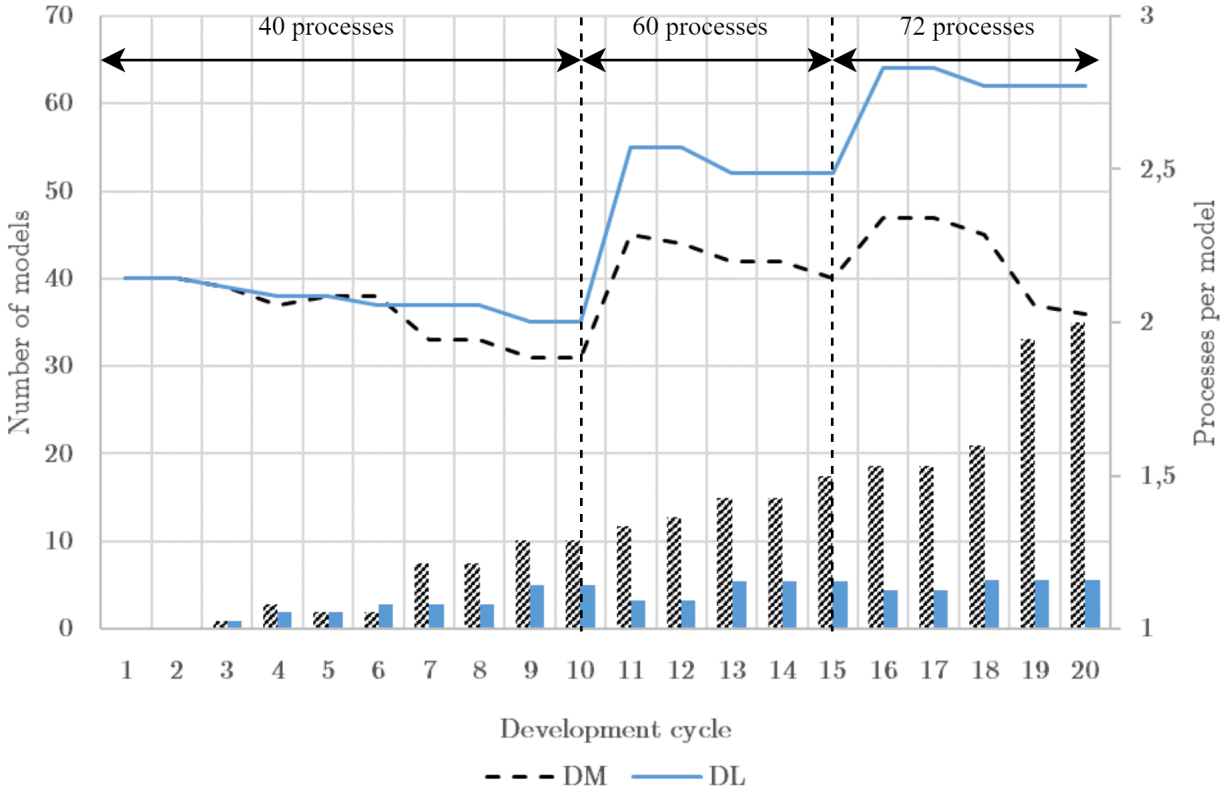


Figure 64: Number of productive models over twenty development cycles as the system is scaled.

The number of processes is *significantly* larger for the pattern matching approach with the spread increasing as the development cycles go on. This can be explained by a combination of two effects. When extending system capacity to an additional process, the standard approach is to train and deploy an unsupervised model and then begin retraining it once feedback from SMEs is available. Additionally, this annotated data can be used to evaluate whether models already in production could be used to monitor the process. If this is the case, it is usually much faster to continue with that existing model rather than training a similar model from scratch. The DL approach requires more labeled data to adapt the normal class than the pattern matching approach. Thus, it is often possible to transfer pattern matching models to new processes even if they have been in production for a relatively short time. The second reason is perhaps the most fundamental advantage of the pattern matching models: they can be applied to different processes as long as the same patterns can be used for classification. For example, the slip-off anomaly discussed in section 6.1 behaves similar for different tightening processes, although their normal state varies significantly. Thus, a single model using a unified pattern library can be used in these processes. In contrast, deep learning models are end-to-end models that do not allow this level of ”modularity”. Both of these effects enable data scientists

to merge multiple pattern matching models into single models that cover multiple processes. This is supported by figure 65 that shows the number of models that monitor multiple processes at the end of the pilot phase. While the majority of models continue to monitor single processes, the pattern matching approach produced many more models that monitor multiple processes.

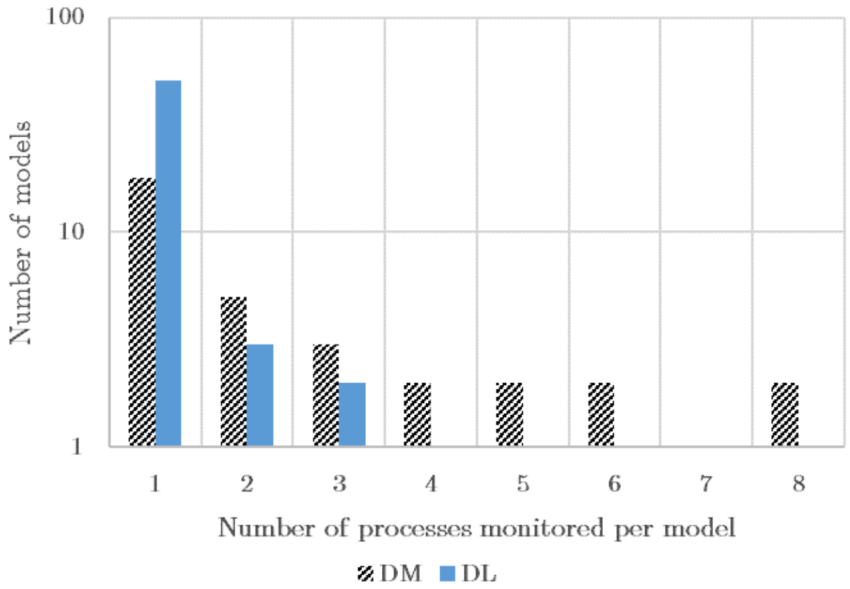


Figure 65: Number of processes monitored per model at the end of the pilot phase.

Lastly, it is important to consider the amount of resources that were expended for (re-)training the models that mad up both systems. For this, the working hours of the data scientist were logged throughout the development cycle. The distribution of the amount of development time for each approach is shown in figure 66. The total time spent on developing the pattern matching models was about 20 percent higher than for the deep learning models. However, these extra resources allowed to decrease the number of models by over 40 percent.

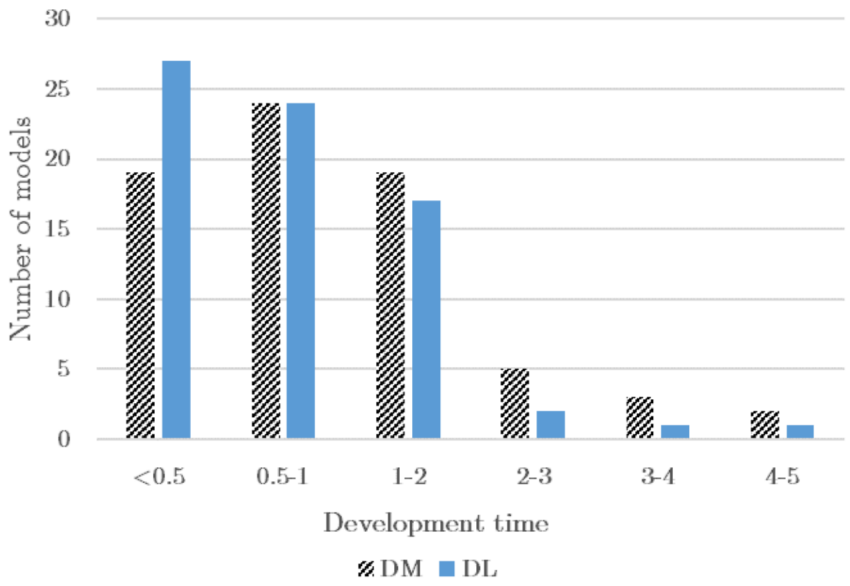


Figure 66: Time spent on model training during the pilot phase.

7 Conclusion

7.1 Critical evaluation of results

Problem statement Vehicle recalls in the automotive industry have been on the rise over the past decades, both in terms of frequency and volume. Their economic burden is immense - the largest recalls have wiped out years of OEM profits and pushed suppliers to the brink of bankruptcy. The cause for this are industry-wide trends that are driven by staunch competition and regulatory oversight that are unlikely to revert in the foreseeable future. The majority of recalls are caused by manufacturing defects. The ability to detect these defects during the manufacturing process and prevent these recalls can be a competitive advantage for manufacturers. In addition to conventional process control systems, manufacturers and researchers are looking to develop intelligent process monitoring systems. The aim of these systems is to analyze the large amounts of process data collected in manufacturing settings and detect patterns that may indicate a defect. Rather than being explicitly programmed to detect known defects, these systems use ML algorithms to *learn* the normal state of a process and detect anomalous behavior that is unknown and would have otherwise gone undetected. These ”known unknowns” - objects that SMEs know they are missing without knowing exactly what to look for, are the cause of many recalls. Despite years of research and numerous pilot applications, the productive use of ML technology for large-scale process monitoring applications in the manufacturing domain has thus far proved evasive. While training models for a limited

pilot environment is relatively easy, scaling this approach to hundreds or even thousands of processes is extremely challenging.

Thesis statement It is widely accepted that moving pilot ML systems into production is extremely challenging. On the one hand, there are obvious difficulties like the training/serving skew of ML models and the scarcity of annotated data, both of which are frequently cited in scientific literature. Additionally, it is increasingly understood that system scalability and operability are fundamental challenges that are often overlooked during the pilot phase. As ML technology has matured and its potential to solve real-world problems in the manufacturing domain has been proven, a growing number of researchers are faced with the question of how to move their pilot systems into production. The need to address these challenges is rapidly gaining in importance. This is evidenced by the number of papers in recent years dedicated to the challenges of deploying, operating and maintaining large-scale productive ML systems. How can the training and retraining process by orchestrated effectively? What is the best way to maximally leverage the limited annotated data that is available? What is a suitable process to repeatedly extend the capacity of the system (scalability)? All of these questions essentially center around the difficulty of managing ML models in production without consuming an unfeasible amount of developer resources that are both costly and difficult to recruit. These challenges are virtually nonexistent during the development phase but dominate during production.

The thesis statement is as follows: the effort required to scale, operate and maintain ML systems is determined by the ease with which models can be managed. This fundamentally depends on the model itself - some models are better suited than others. However, model selection is unsystematic in practice. Sub-optimal model selection results in increased effort for model management. As the system is scaled, this may consume a rapidly growing amount of developer resources until the complexity overwhelms the development team. The system quickly becomes unfeasible, often before becoming fully operational. This is the reason for the low adoption rate of ML technology in manufacturing, that has remained stubbornly low for many years. Leveraging ML technology for process monitoring requires ML algorithms that can be effectively orchestrated and used to train and adapt models that can be easily maintained and extended on a system level.

The impulse for this thesis was the experience gained during years of applied ML research in an industrial settings in the automotive manufacturing industry. The challenges associated with scaling a pilot system and moving it into production were constantly recurring issues. The apparent difficulty of ML practitioners, both within and outside the manufacturing domain, showed that this challenge was not exclusive to the manufacturing setting but systemic across industries that needed to be overcome.

Goal of thesis The goal of this thesis is to provide ML practitioners in the manufacturing domain with a set of tools to overcome the challenges of scaling a continuously adaptive system for anomaly detection as well as operating and maintaining that system at scale. Unsupervised anomaly detection is fundamentally a threshold problem that cannot be inferred from the data alone. Instead, the goal is to rapidly transform the anomaly detector into a one-class classifier based on user feedback. The core of the work focused on applied ML research and development of ML algorithms that are able to effectively leverage scarcely labeled highly imbalanced data to train models that are easy to debug and maintain. Following the Design Science Research methodology, this development incorporated state-of-the-art algorithms and design principles and relied on rigorous evaluation of the developed algorithms under real-world conditions to ensure the generalizability of results.

Summary of results Based on a comprehensive review of existing literature, two distinct modeling approaches were identified as particularly promising. The first are deep learning models, that have been shown to outperform most other ML models for a wide range of anomaly detection applications. The second are time series data mining models that classify an object based a library of characteristic patterns, similar to template-based computer vision systems. This approach combines data mining algorithms to recognize patterns in the data and conventional ML models to classify objects based on their pattern-based feature representation. Both model types offer distinct relative advantages and were therefore pursued in parallel. A comparative evaluation of these approaches was conducted in the form of a large-scale evaluation phase that included training, deployment and adaptation of a large number of models in a real-world manufacturing setting. The number of processes monitored during the evaluation phase can be considered large-scale, compared to published results of comparable systems in the manufacturing domain [27, 139, 140]. The findings are summarized below:

- Both the average time to deployment and the average time spent on model retraining was

approximately 20 percent higher for DM models compared to DL models. On the other hand, the number of models required for monitoring the production system was almost 50 percent lower. Essentially, the DM approach allows trading off a short-term increase in resources required for model training for a long-term decrease in resources required for system operation.

- The majority of DL models only ever monitored a single process. In comparison, DM models monitored an average of two processes with up to eight monitored by a single model.

- In settings where only scarcely labeled data is available (as is often the case in real-world settings) it is difficult to rely on hyperparameter tuning as the sole hit-or-miss training strategy as it risks overfitting the model. The ability to analyze what the model has learned can help guide this process. The pattern libraries used by the DM models for classifying the time series objects provide an intuitive understanding of what the model is looking for in the data.

All things considered, time series data mining models have proven to be the more scalable approach in *relative* terms. The 20-week real-world evaluation showed, that it is (comparatively) much easier to scale single data mining models to multiple processes. The lower number of models in production reduces the overall complexity of the ML system and simplifies its operation. Additionally, scaling a model across multiple processes increases the amount of annotated training data per model, resulting in more generalizable models and making it possible to transfer these pre-trained models to new processes (avoiding the ”cold-start problem”). These effects compound as the number of processes in the ML system increases. The added complexity of managing the data dependencies of these scaled models can be mitigated through the use of ML Ops software systems that orchestrate the training process. Although more time was spent on managing data mining models compared to the deep learning models, this is not due to a weakness of the former. On the contrary, data scientists *chose* to spend more time on these models because it made sense for them to do so. Most of that time was spent analyzing the extracted patterns and retracing the decision process of these model to introduce targeted changes to the hyperparameters during model training. In contrast, associating the latent features of the DL models with characteristic features in the input data was not done in practice, even though this was shown to be technically possible in section 5. Instead, data scientists relied solely on

the high-level trial-and-error hyperparameter tuning that is often used for black-box DL models.

It would be disingenuous to generalize this conclusion to *all* possible data mining and deep learning approaches that could be used for *any* process monitoring application in manufacturing. The research in this thesis is limited to the two approaches described in sections 4 and 5 and a particular setting in the automobile manufacturing industry. However, since few restrictions were placed on the manufacturing data itself and the applied research methodology was centered around iterative development on benchmark data sets and field-testing under real-world conditions, the insights are at least relevant to similar settings. The primary reason behind the success of the pattern matching approach is that it combines the strengths of DM algorithms for pattern recognition and ML algorithms for learning the decision boundary. There is evidence in scientific literature that supports this conclusion: ”In recent years there has been an explosion of deep learning work on anomaly detection, [...]. However, we feel that there is currently little evidence presented that the complexity of these approaches is warranted. Recall that for the most part we can reproduce or improve upon these results [...] using a method that is, by any reasonable standard, an order of magnitude simpler” [242]. Interestingly, successful large-scale productive ML systems in other real-world domains also make use of a hybrid DM/ML framework [144, 41] (albeit without the focus on time series data that is central to the manufacturing domain and this thesis).

7.2 Research contribution

This subsection discusses the research artifacts, their contribution to the current state-of-the-art and their implications for researchers and practitioners in the manufacturing domain and beyond. They can be categorized into artifact types according to the typology proposed by Offermann et al. [66]. The artifact types relevant to this research are (1) algorithms: an executable description of a system behavior, (2) guidelines: a generlized suggestion about system development, (3) patterns: generalized system design elements that can be used as a blueprint for creating a system design, and (4) system design: an IT-system that comprises the system architecture, processes and human interactions on any level of granularity. The artifacts together with their categorization are as follows:

- An improvement of a state-of-the-art supervised pattern extraction algorithm that enables the extraction of patterns with minimal information redundancy from highly imbalanced

data (algorithm artifact). This is central to the hybrid DM/ML pattern matching approach and relevant to virtually all classification problems where only a limited amount of annotated data is available. In many practical settings, the cost of manually labeling more data may be prohibitively expensive or more data of the target class may simply not be available. Instead, the algorithm redirects these efforts towards extracting more effective features from the data, following the maxim ”better data beats more data” [243].

- An approach for constructing and extending pattern matching models for one-class time series classification. The approach is based on existing DM algorithms that are used to construct and extend a library of patterns based on user feedback (algorithm artifact).

- An approach to constructing DL models for one-class classification of time series objects. The central element is the loss function that (1) ensures a normally distributed feature space that simplifies the threshold problem (highly desirable to simplify the threshold problem of unsupervised anomaly detection) and improves model interpretability and (2) allows to continuously retrain the one-class model with annotated data that becomes available over time (algorithm artifact). Additionally, design considerations are provided regarding the ANN architecture of DL algorithms for time series analysis (pattern artifact).

- A similarity search system for the high-recall retrieval of objects from large time series databases. The system adapts the search query based on used-provided feedback on the search results (system design artifact). It enables users to explore and annotate otherwise intractable data sets.

- A MLOps software system for training, deploying and managing a large number of ML models in production (system design artifact). The system implements a proposed graph-based method for model life cycle management and includes web-based front-ends for model management by ML practitioners and data annotation by SMEs.

The source code of all research artifacts has been published along with peer-reviewed academic papers [229, 7, 6, 167]. The model life cycle management system that was used for the real-world evaluation described in section 6.4 was packaged into a stand-alone software and open-sourced [244] and can be used with any conventional ML algorithm. This work aims to go beyond the presentation of new algorithms that improve model performance on benchmark data sets or the use of out-of-the-box algorithms in small pilot systems to showcase the potential of

ML technology for certain applications. Instead, it takes a holistic view at ensuring system scalability and operability by developing end-to-end approaches to model management. It is increasingly understood within the scientific community, that these system-level challenges are behind the widespread difficulty to transfer promising pilot applications into productive systems that add business value.

7.3 Outlook

This section discusses directions for further research to improve upon the results of this work and provides concluding comments on the applicability of the framework.

Firstly, more research should be directed towards further improving system scalability. Although the proposed framework is sufficient for developing and operating an ML system of significant size, the validation phase showed, that there is still need for significant improvement. To scale system capacity, the framework relies on a two (alternative) approaches for initializing and continuously retraining one-class models for anomaly detection. This requires an effective retraining process that avoids model degradation after retraining. Although the training process can be largely automated, manual intervention is often necessary. A feasible alternative may be to separate the anomaly detection and classification tasks by using simple, static models for anomaly detection and training classification models for all annotated data. This would shift the training objective away from one-class classification towards conventional binary classification. Provided effective orchestration of the training pipelines, this could allow a vastly simpler system. ”It is easier to improve a system that provides adequate performance at scale than to scale a system that was not designed to grow” [144].

Secondly, researchers should work on additional measures to improve the long-term operability of the system. The starting point should be a critical examination of the approaches proposed in this thesis and their suitability for changing requirements as the anomaly detection system matures. After the system has been in production for some time, understanding of the normal class will become well-established among SMEs and ML practitioners. As the frequency of annotating data and retraining models decreases, it makes sense to construct rule-based models that formalize this knowledge. This would make it possible to further reduce model complexity. The pattern libraries used by the DM models for classifying objects could serve as an ideal starting point for the construction of these models - this automatic systematizing of domain

knowledge gathered from user feedback is essentially a free by-product of the DM approach.

The work presented in this thesis provides ML practitioners in the manufacturing domain with a framework for building large-scale anomaly detection systems for time series data. This includes dedicated ML algorithm, tools to rapidly annotate large amounts of data as well as a software system for training and deploying ML models and monitoring their performance in production. It is likely that some changes to the algorithms may be necessary to make the framework applicable for a particular setting - however, it is encouraging, to know that researchers in other domains have followed a similar path of building ”large-scale data mining systems” [144, 41] to the problem at hand. Perhaps a path towards a systematic approach to transferring ML technology to the problem of anomaly detection in manufacturing is discernible, after all.

References

[1] E. Siegel, Models are rarely deployed: An industry-wide failure in machinelearning leadership (2022).

[2] N. Bauer, L. Stankiewicz, M. Jastrow, D. Horn, J. Teubner, K. Kersting, J. Deuse, C. Weihs, Industrial data science: Developing a qualification concept for machine learning in industrial production (2018). doi:10.5445/KSP/1000087327/27.

[3] A. Hevner, A three cycle view of design science research, Scandinavian Journal of Information Systems 19 (2) (2007) 87–92.

[4] E. Keogh, M. Pazzani, Scaling up dynamic time warping for datamining applications, in: 6th ACM SIGKDD international conference on Knowledge discovery and data mining, 2000, pp. 285–289.

[5] A. Mueen, E. Keogh, Q. Zhu, S. Cash, B. Westover, Exact discovery of time series motifs, Proceedings of the ... SIAM International Conference on Data Mining. SIAM International Conference on Data Mining 2009 (2009) 473–484. doi:10.1137/1.9781611972795.41.

[6] T. Schlegl, S. Schlegl, A. Sciberras, N. West, J. Deuse, Margin-based greedy shapelet search for robust time series classification of imbalanced data, in: IEEE International Conference on Big Data, IEEE, 2021, pp. 5266–5274.

[7] T. Schlegl, S. Schlegl, A. Sciberras, Greedy shapelet search (2021).
URL https://github.com/papelero/greedy_shapelets

[8] T. Schlegl, S. Schlegl, Adaptive similarity search for large time series archives (2021).
URL https://github.com/papelero/adaptive_search

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.

[10] T. E. Hedrick, Bickman, Leonard, Rog, Debra, Applied Research Design: A Practical Guide, SAGE Publications, 1993.

[11] U.S. National Highway Traffic Safety Administration, Annual recall report (2020).

[12] M. Held, A. Marian, J. Reaves, The auto dinsutry's growing recall problem - and how to fix it.

[13] K. Hill, D. Maranger Menk, Cooper Adam, Contribution of the automotive industry to the economies of all fifty states and the united states.

[14] H. Bates, M. Holweg, M. Lewis, N. Oliver, Motor vehicle recalls: Trends, patterns and emerging issues, Omega 35 (2) (2007) 202–210. doi:10.1016/j.omega.2005.05.006.

[15] E. Muralidharan, H. Bapuji, Product recalls: A review of literature (2009).

[16] K. Ahsan, Trend analysis of car recalls: Evidence from the us market, International Journal of Managing Value and Supply Chains 4 (4) (2013) 1–16. doi:10.5121/ijmvsc.2013.4401.

[17] K. Ahsan, I. Gunawan, Analysis of product recalls: Identification of recall initiators and causes of re-call, Operations and Supply Chain Management: An International Journal (2014) 97–106doi:10.31387/oscm0180115.

[18] C.-F. Chi, D. Sigmund, M. O. Astardi, Classification scheme for root cause and failure modes and effects analysis (fmea) of passenger vehicle recalls, Reliability Engineering & System Safety 200 (2020) 106929. doi:10.1016/j.ress.2020.106929.

[19] H.-J. Lenz, P.-T. Wilrich (Eds.), Frontiers in Statistical Quality Control 7, Physica-Verlag HD, Heidelberg, 2004. doi:10.1007/978-3-7908-2674-6.

[20] D. Pelleg, A. Moore, Active learning for anomaly and rare-category detection, in: L. Saul, Y. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems 17, The MIT Press, 2005.

[21] Y. Pan, Heading toward artificial intelligence 2.0, Engineering 2 (4) (2016) 409–413. doi:10.1016/J.ENG.2016.04.018.

[22] J. Grotepass, J. Diemer, Artificial intelligence in industrial automation.

[23] J. Bughin, E. Hauzan, S. Ramaswamy, M. Chui, T. Alles, P. Dahlstroem, N. Henke, M. Trench, Artificial intelligence - the next digital frontier: Discussion paper.

[24] The global ai agenda.

[25] D. Lieber, M. Stolpe, B. Konrad, J. Deuse, K. Morik, Quality prediction in interlinked manufacturing processes based on supervised & unsupervised machine learning, in: P. Cunha (Ed.), 46th CIRP Conference on Manufacturing Systems, Vol. 7, Elsevier Procedia, 2013, pp. 193–198.

[26] R.-J. Hsieh, J. Chou, C.-H. Ho, Unsupervised online anomaly detection on multivariate sensing time series data for smart manufacturing, in: 12th Conference on Service-Oriented Computing and Applications (SOCA), IEEE, 2019, pp. 90–97.

[27] N. Laptev, S. Amizadeh, I. Flint, Generic and scalable framework for automated time-series anomaly detection, in: L. Cao, C. Zhang, T. Joachims, G. Webb, D. D. Margineantu, G. Williams (Eds.), 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 15), ACM Press, New York, New York, USA, 2015, pp. 1939–1947.

[28] B. Lindemann, F. Fesenmayr, N. Jazdi, M. Weyrich, Anomaly detection in discrete manufacturing using self-learning approaches, Procedia CIRP 79 (2019) 313–318.

[29] G. A. Susto, M. Terzi, A. Beghi, Anomaly detection approaches for semiconductor manufacturing, Procedia Manufacturing 11 (2017) 2018–2024.

[30] X. Wang, J. Lin, N. Patel, M. Braun, A self-learning and online algorithm for time series anomaly detection, with application in cpu manufacturing, in: S. Mukhopadhyay, C. Zhai, E. Bertino, F. Crestani, J. Mostafa, J. Tang, L. Si, X. Zhou, Y. Chang, Y. Li, P. Sondhi (Eds.), 25th ACM International on Conference on Information and Knowledge Management (CIKM 2016), ACM Press, New York, NY, USA, 2016, pp. 1823–1832.

[31] L. Baier, F. Jöhren, S. Seebacher, Challenges in the deployment and operation of machine learning in practice, in: Proceedings of the 27th European Conference on Information Systems (ECIS), 2019.

[32] T. Davenport, K. Malone, Deployment as a critical business data science discipline, Harvard Data Science Review 3 (1) (2021). doi:10.1162/99608f92.90814c32.

[33] 2020 state of enterprise machine learning.

[34] S. Ransbotham, S. Khodabandeh, D. Kiron, F. Candelon, M. Chu, B. Lafountain, Expanding ai's impact with organizational learning: Finding from the 2020 artificial intelligence global executive study and research project (2020).

[35] D. E. Stokes, Pasteur's quadrant: Basic Science and Technological Innovation, Brookings Institutional Press, 1997.

[36] Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, Dan Dennison, Hidden technical debt in machine learning systems (2015).

[37] D. Dahlmeier, On the challenges of translating nlp research into commercial products, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2017, pp. 92–96.

[38] M. Kim, T. Zimmermann, R. DeLine, A. Begel, Data scientists in software teams: State of the art and challenges, IEEE Transactions on Software Engineering (2017) 1024–1038.

[39] S. Baškarada, A. Koronios, Unicorn data scientist: the rarest of breeds, Program 51 (1) (2017) 65–74. doi:10.1108/PROG-07-2016-0053.

[40] K. Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, Xiaodong Wang, Applied machine learning at facebook: A datacenter infrastructure perspective, 2018 IEEE International Symposium on High Performance Computer Architecture (2018).

[41] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, Y. Zhou, Detecting adversarial advertisements in the wild, in: C. Apte, J. Ghosh, P. Smyth (Eds.), Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11, ACM Press, New York, New York, USA, 2011, p. 274. doi:10.1145/2020408.2020455.

[42] S. Tata, A. Popescul, M. Najork, M. Colagrosso, J. Gibbons, A. Green, A. Mah, M. Smith, D. Garg, C. Meyer, R. Kan, Quick access: Building a smart experience for google drive, in: S. Matwin, S. Yu, F. Farooq (Eds.), Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2017, pp. 1643–1651. doi:10.1145/3097983.3098048.

[43] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: S. Sen, W. Geyer, J. Freyne, P. Castells (Eds.), Proceedings of the 10th ACM Conference on Recommender Systems, ACM, New York, NY, USA, 2016, pp. 191–198. doi:10.1145/2959100.2959190.

[44] B. Williams, Automated Vehicles and MaaS: Removing the Barriers, John Wiley & Sons, 2021.

[45] H. Kallstrom, Raw materials – the biggest cost driver in the auto industry.

[46] A. Shyam, Automakers to benefit from softening steel prices, duty cuts on fuels.

[47] R. M. Cuenca, L. L. Gaines, A. D. Vyas, Evaluation of electric vehicle production and operating costs.

[48] A. Vyas, D. Santini, R. Cuenca, Comparison of indirect cost multipliers for vehicle manufacturing.

[49] R. D. Banker, S. M. Datar, D. Zweig, Software complexity and maintainability, in: J. I. DeGross, J. C. Henderson, B. R. Konsynski (Eds.), Proceedings of the tenth international conference on Information Systems - ICIS '89, ACM Press, New York, New York, USA, 1989, pp. 247–255. doi:10.1145/75034.75056.

[50] A. Paleyes, R.-G. Urma, N. Lawrence, Challenges in deploying machine learning: a survey of case studies, in: The ML-Retrospectives, Surveys & Meta-Analyses Workshop, NeurIPS 2020, 2020.

[51] S. Sabbeh, Machine-learning techniques for customer retention: A comparative study, International Journal of Advanced Computer Science and Applications 9 (2) (2018). doi:10.14569/IJACSA.2018.090238.

[52] F. Gharibian, A. Ghorbani, Comparative study of supervised machine learning techniques for intrusion detection, in: Fifth Annual Conference on Communication Networks and Services Research (CNSR '07), IEEE, 2007, pp. 350–358. doi:10.1109/CNSR.2007.22.

[53] J. Hagenauer, M. Helbich, A comparative study of machine learning classifiers for modeling travel mode choice, Expert Systems with Applications 78 (2017) 273–282. doi:10.1016/j.eswa.2017.01.057.

[54] C. Hill, R. Bellamy, T. Erickson, M. Burnett, Trials and tribulations of developers of intelligent systems: A field study, in: IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE, 2016.

[55] R. Phonson, B. Foote, Designing reusable classes, Journal of Object-Oriented Programming 1 (2) (1988) 22–35.

[56] L. Cao, Data science: A comprehensive overview, ACM Computing Surveys 50 (3) (2017) 1–42. doi:10.1145/3076253.

[57] F. Provost, R. Kohavi, Guest editors' introduction: On applied research in machine learning, Machine Learning 30 (1998) 127–132.

[58] C. Zeng, S. Li, Q. Li, J. Hu, J. Hu, A survey on machine reading comprehension—tasks, evaluation metrics and benchmark datasets, Applied Sciences 10 (21) (2020) 7640. doi:10.3390/app10217640.

[59] J. Dunietz, G. Burnham, A. Bharadwaj, O. Rambow, J. Chu-Carroll, D. Ferrucci, To test machine comprehension, start by defining comprehension, in: D. Jurafsky, J. Chai, N. Schluter, J. Tetreault (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, 2020, pp. 7839–7859. doi:10.18653/v1/2020.acl-main.701.

[60] H. Kerner, Too many ai researchers think real-world problems are not relevant: The community's hyperfocus on novel methods ignores what's really important., MIT Technology Review (2020).

[61] Y. Bengio, Time to rethink the publication process in machine learning (2020).

[62] J. Dunietz, The field of natural language processing is chasing the wrong goal, MIT Technology Review (2020).

[63] Z. C. Lipton, J. Steinhardt, Troubling trends in machine learning scholarship: Some ml papers suffer from flaws that could mislead the public and stymie future research (2019).

[64] H. Kara, K. Gergen, M. Gergen, Creative research methods in the social sciences: A practical guide, Policy Press, Bristol, 2015.

[65] A. Hevner, S. March, J. Park, S. Ram, Design science in information systems research, Management Information Systems Quarterly 28 (1) (2004) 75–105.

[66] P. Offermann, S. Blom, M. Schönherr, U. Bub, Artifact types in information systems design science – a literature review, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, R. Winter, J. L. Zhao, S. Aier (Eds.), Global Perspectives on Design Science Research, Vol. 6105 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 77–92.

[67] A. Dresch, D. P. Lacerda, J. A. V. Antunes Jr, Design Science Research, Springer International Publishing, Cham, 2015.

[68] A. Hevner, S. Chatterjee, Design Research in Information Systems: Theory and Practice, Springer, 2010.

[69] J. Iivari, A paradigmatic analysis of information systems as a design science, Scandinavian Journal of Information Systems 19 (2) (2007) 39–64.

[70] K. Pfeffers, M. Rothenberger, T. Tuunanen, R. Vaezi, Design science research evaluation, in: International Conference on Design Science Research in Information Systems, Springer, 2012, pp. 398–410.

[71] K. M. Adams, Nonfunctional Requirements in Systems Analysis and Design, Vol. 28, Springer International Publishing, Cham, 2015. doi:10.1007/978-3-319-18344-2.

[72] J. Goodenough, C. Weinstock, On system scalability: Performance-critical systems.

[73] E. Weyuker, A. Avritzer, A metric to predict software scalability, in: D. Williams (Ed.), Proceedings of the 8th IEEE Symposium on Software Metrics, IEEE, 2002.

[74] A. Bagnall, J. Lines, A. Bostrom, J. Large, E. Keogh, The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances, Data mining and knowledge discovery 31 (3) (2017) 606–660. doi:10.1007/s10618-016-0483-9.

[75] J. Zakaria, A. Mueen, E. Keogh, Clustering time series using unsupervised-shapelets, in: 2012 IEEE 12th International Conference on Data Mining, IEEE, 2012, pp. 785–794. doi:10.1109/ICDM.2012.26.

[76] M. Ram (Ed.), Safety and Reliability Modeling and its Applications: Reliability and maintainability of safety instrumented system, Advances in Reliability Science, Elsevier, 2021.

[77] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 1–37.

[78] J. Wang, B. Guo, L. Chen, Human-in-the-loop machine learning: A macro-micro perspective (2022).

[79] S. Amershi, M. Cakmak, W. B. Knox, T. Kulesza, Power to the people: The role of humans in interactive machine learning, AI Magazine 35 (4) (2015) 105–120. doi:10.1609/aimag.v35i4.2513.

[80] J. J. Dudley, P. O. Kristensson, A review of user interface design for interactive machine learning, ACM Transactions on Interactive Intelligent Systems 8 (2) (2018) 1–37. doi:10.1145/3185517.

[81] S. Teso, K. Kersting, "why should i trust interactive learners?" explaining interactive queries of classifiers to users. doi:NIPS.
URL http://arxiv.org/pdf/1805.08578v1

[82] D. V. Carvalho, E. M. Pereira, J. S. Cardoso, Machine learning interpretability: A survey on methods and metrics, Electronics 8 (8) (2019) 832. doi:10.3390/electronics8080832.

[83] Y. Lou, R. Caruana, J. Gehrke, Intelligible models for classification and regression, in: 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD 12), ACM, 2012.

[84] S. Chakraborty, R. Tomsett, R. Raghavendra, D. Harborne, M. Alzantot, F. Cerutti, Srivastava, Mani, A. Preece, S. Julier, R. Rao, T. Kelley, D. Braines, M. Sensoy, C. Willis, P. Gurram, Interpretability of deep learning models: A survey of results, in: Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing, UIC-ATC, IEEE, 2017.

[85] Q.-s. Zhang, S.-c. Zhu, Visual interpretability for deep learning: a survey, Frontiers of Information Technology & Electronic Engineering 19 (1) (2018) 27–39.

[86] A. Chandola, A. Banarjee, V. Kumar, Anomaly detection: A survey, ACM Computing Surveys (3) (2009).

[87] C. Perlich, B. Dalessandro, T. Raeder, O. Stitelman, F. Provost, Machine learning for targeted dis-play advertising: transfer learning in action, Machine Learning 95 (1) (2014) 103–127. doi:10.1007/s10994-013-5375-2.

[88] C.-A. Brust, C. Käding, J. Denzler, Active and incremental learning with weak supervision, KI - Künstliche Intelligenz 34 (2) (2020) 165–180. doi:10.1007/s13218-020-00631-4.
URL http://arxiv.org/pdf/2001.07100v1

[89] B. Settles, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2012.

[90] S. Das, W.-K. Wong, T. Dietterich, A. Fern, A. Emmott, Incorporating expert feedback into active anomaly discovery, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016.

[91] C. Rudin, K. L. Wagstaff, Machine learning for science and society, Machine Learning 95 (1) (2014) 1–9.

[92] J. Shieh, E. Keogh, isax: Indexing and mining terabyte sized time series, in: Proceedings of the 14th ACMKDD International Conference on Knowledge Discovery & Data Mining, 2008, pp. 623–632.

[93] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, E. Keogh, Generalizing dtw to the multi-dimensional case requires an adaptive approach, Data mining and knowledge discovery 31 (1) (2017) 1–31. doi:10.1007/s10618-016-0455-0.

[94] T. Giorgino, Computing and visualizing dynamic time warping alignments in r: The dtw package, Journal of Statistical Software 31 (7) (2009). doi:10.18637/jss.v031.i07.

[95] I. P. Androulakis, Selecting maximally informative genes, Computers & Chemical Engineering 29 (3) (2005) 535–546. doi:10.1016/j.compchemeng.2004.08.037.

[96] G. E. Batista, B. Campana, E. Keogh, Classification of live moths combining texture, color and shape primitives, in: 9th International Conference on Machine Learning and Applications, IEEE, 2010, pp. 903–906.

[97] S. Gharghabi, S. Imani, A. Bagnall, A. Darvishzadeh, E. Keogh, Matrix profile xii: Mpdist: A novel time series distance measure to allow data mining in more challenging scenarios, in: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 965–970. doi:10.1109/ICDM.2018.00119.

[98] M. Riesenhuber, T. Poggio, Models of object recognition, Nature Neuroscience (3) (2000) 1199–1204.

[99] W. Shugen, Framework of pattern recognition model based on the cognitive psychology, Geo-spatial Information Science 5 (2) (2002).

[100] F. Bischoff, F. Cancino, J. Green, T. Marrs, A. Ouyang, A. van Benschoten, The matrix profile foundation (2020).
URL https://matrixprofile.org/

[101] H. A. Dau, E. Keogh, Matrix profile v: A generic technique to incorporate domain knowledge into motif discovery, in: S. Matwin, S. Yu, F. Farooq (Eds.), Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, 2017, pp. 125–134. doi:10.1145/3097983.3097993.

[102] Y. Zhu, A. Mueen, E. Keogh, Matrix profile ix: Admissible time series motif discovery with missing data, IEEE Transactions on Knowledge and Data Engineering (2018).

[103] K. Chakrabarti, E. Keogh, S. Mehotra, M. Pazzani, Locally adaptive dimensionality reduction for indexing large time series databases, ACM Transactions on Database Systems 27 (2) (2002) 188–228.

[104] I. Assent, R. Krieger, F. Afschari, T. Seidl, The ts-tree: Efficient time series search and retrieval (2008).

[105] M. Christ, A. Kempa-Liehr, M. Feindt, Distributed and parallel time series feature extraction for industrial big data applications, Neurocomputing (2017).

[106] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh, Querying and mining of time series data: Experimental comparison of representations and distance measures, in: P. Buneman, B. C. Ooi, K. Ross, G. Weber (Eds.), Proceedings of the VLDB Endowment, ACM, New York, NY, United States, 2008, pp. 1542–1552.

[107] G. E. Batista, X. Wang, E. J. Keogh, A complexity-invariant distance measure for time series, in: B. Liu, H. Liu, C. Clifton, T. Washio, C. Kamath (Eds.), Proceedings of the 2011 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2011, pp. 699–710. doi:10.1137/1.9781611972818.60.

[108] M. Feurer, J. T. Springberg, A. Klein, M. Blum, K. Eggensperger, F. Hutter, Efficient and robust automated machine learning, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, MIT Press, Cambrdige, MA, USA, 2015, pp. 2755–2763.

[109] J. Deuse, M. Wiegand, K. Weisner, Continuous process monitoring through ensemble-based anomaly detection, in: N. Bauer, K. Ickstadt, K. Lübke, G. Szepannek, H. Trautmann, M. Vichi (Eds.), Applications in Statistical Computing: Studies in Classification, Data Analysis, and Knowledge Organization, Springer Nature Switzerland, Cham, Switzerland, 2019, pp. 289–304.

[110] T. Ergen, S. S. Kozat, Unsupervised anomaly detection with lstm neural networks, IEEE transactions on neural networks and learning systems 31 (8) (2020) 3127–3141. doi:10.1109/TNNLS.2019.2935975.

[111] G. Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel, Deep learning for anomaly detection: A review (2020).

[112] D. Kim, H. Yang, M. S. Chung, Squeezed convolutional variational autoencoder for unsupervised anomaly detection in edge device industrial internet of things, in: 4th International Conference on Information and Computer Technologies (ICICT), 2018, pp. 67–71.

[113] J. Pereira, M. Silveira, Unsupervised anomaly detection in energy time series data using variational recurrent autoencoders with attention, in: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE, 2018, pp. 1275–1282. doi:10.1109/ICMLA.2018.00207.

[114] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, T. Soderstrom, Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding 60 (2018) 387–395. doi:10.1145/3219819.3219845. URL http://arxiv.org/pdf/1802.04431v3

[115] T. Kieu, B. Yang, C. Guo, C. S. Jensen, Outlier detection for time series with recurrent autoencoder ensembles, in: T. Eiter, S. Kraus (Eds.), Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, California, 2019, pp. 2725–2732. doi:10.24963/ijcai.2019/378.

[116] J. Liu, J. Guo, P. Orlik, M. Shibata, D. Nakahara, S. Mii, M. Takac, Anomaly detection in manufacturing systems using structured neural networks, in: Y. Liu, Y. Wang (Eds.), 13th World Congress on Intelligent Control and Automation (WCICA 2018), IEEE, 2018, pp. 175–180.

[117] T. Schlegl, S. Schlegl, J. Deuse, Detektion von anomalien in automatisierten schraubprozessen: Erprobung von autoencodern zum erlernen des normalzustandes von drehmomentverläufen, Zeitschrift für wirtschaftlichen Fabrikbetrieb (5) (2020).

[118] F. Hohman, H. Park, C. Robinson, D. H. Chau, Summit: Scaling deep learning interpretability by visualizing activation and attribution summarizations, IEEE Transactions on Visualization and Computer Graphics (2017).

[119] M. Kahng, P. Andrews, A. Kalro, D. H. Chau, Visual exploration of industry-scale deep neural network models, IEEE Transactions on Visualization and Computer Graphics (2019).

[120] J. Gao, X. Song, Q. Wen, P. Wang, L. Sun, H. Xu, Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks. doi:paper.
URL http://arxiv.org/pdf/2002.09545v2

[121] M. Munir, S. A. Siddiqui, M. A. Chattha, A. Dengel, S. Ahmed, Fusead: Unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models, Sensors (Basel, Switzerland) 19 (11) (2019). doi:10.3390/s19112451.

[122] M. Munir, S. A. Siddiqui, A. Dengel, S. Ahmed, Deepant: A deep learning approach for unsupervised anomaly detection in time series, IEEE Access 7 (2019) 1991–2005.

[123] X. Zhou, Y. Hu, W. Liang, J. Ma, Q. Jin, Variational lstm enhanced anomaly detection for industrial big data, IEEE Transactions on Industrial Informatics 17 (5) (2021) 3469–3477. doi:10.1109/TII.2020.3022432.

[124] M. Maggipinto, A. Beghi, G. Antonio Susto, A deep learning-based approach to anomaly detection with 2-dimensional data in manufacturing, in: 17th IEEE International Conference on Industrial Informatics (INDIN19), IEEE, 2019, pp. 187–192.

[125] N. Madiraju, S. Sadat, D. Fisher, H. Karamabadi, Deep temporal clustering: Fully unsupervised learning of time-domain features, in: Y. Bengio, Y. LeCun (Eds.), 6th International conference on Learning Representations (ICLR), 2018.

[126] S. Zhao, J. Song, S. Ermon, Learning hierarchical features from deep generative models, in: Proceedings of the 34th International Conference on Machine Learning, ACM, 2017, pp. 4091–4099.

[127] L. Li, J. Yan, H. Wang, Y. Jin, Anomaly detection of time series with smoothness-inducing sequential variational auto-encoder. doi:IEEE.
URL http://arxiv.org/pdf/2102.01331v1

[128] A. Nguyen, J. Yosinski, J. Clune, Understanding neural networks via feature visualization: A survey, in: W. Samek, G. Montavon, A. Vedaldi, L. Hansen, K. Müller (Eds.), Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, Springer, Cham, Switzerland, 2019, pp. 55–76.

[129] Q. Zhang, J. Wu, H. Yang, Y. Tian, C. Zhang, Unsupervised feature learning from time series, in: 25th International Joint Conference on Artificial Intelligence (IJCAI-16), IEEE, 2016, pp. 2322–2328.

[130] J. Pereira, Margarida Silveira, 2019 IEEE International Conference on Big Data and Smart Computing (BigComp): Proceedings : 27 February-2 March 2019, Kyōto, Japan, IEEE, Piscataway, NJ, 2019.
URL https://ieeexplore.ieee.org/servlet/opac?punumber=8671661

[131] P. Perera, V. Patel, Learning deep features for one-class classification, IEEE Transactions on Image Processing (2019).

[132] I. Golan, R. El-Yaniv, Deep anomaly detection using geometric transformations.
URL http://arxiv.org/pdf/1805.10917v2

[133] H. Hemati, M. Schreyer, D. Borth, Continual learning for unsupervised anomaly detection in continuous auditing of financial accounting data. doi:AAAI.
URL http://arxiv.org/pdf/2112.13215v2

[134] S. Ahmad, A. Lavin, S. Purdy, Z. Agha, Unsupervised real-time anomaly detection for streaming data, Neurocomputing 262 (2017) 134–147. doi:10.1016/j.neucom.2017.04.070.

[135] S. Saurav, P. Malhotra, V. TV, N. Gugulothu, L. Vig, P. Agarwal, G. Shroff, Online anomaly detection with concept drift adaptation using recurrent neural networks, in: S. Ranu, N. Ganguly, R. Ramakrishnan, S. Sarawagi, S. Roy (Eds.), Proceedings of the ACM India Joint International Conference on Data Science and Management of Data (COMAD18), ACM Press, New York, New York, USA, 2018, pp. 78–87.

[136] A. Stocco, P. Tonella, Towards anomaly detectors that learn continuously, in: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2020, pp. 201–208. doi:10.1109/ISSREW51248.2020.00073.

[137] V. M. Souza, R. G. Rossi, G. E. Batista, S. O. Rezende, Unsupervised active learning techniques for labeling training sets: An experimental evaluation on sequential data, Intelligent Data Analysis 21 (5) (2017) 1061–1095. doi:10.3233/IDA-163075.

[138] K. Doshi, Y. Yilmaz, Continual learning for anomaly detection in surveillance videos. doi:CVPR.
URL http://arxiv.org/pdf/2004.07941v1

[139] G. Chen, Z. Ge, Hierarchical bayesian network modeling framework for large-scale process monitoring and decision making, IEEE Transactions on Control Systems Technology 28 (2) (2020) 671–679. doi:10.1109/TCST.2018.2882562.

[140] N. Davis, G. Raina, K. Jagannathan, A framework for end-to-end deep learning-based anomaly detection in transportation networks, Transportation Research Interdisciplinary Perspectives 5 (2020).

[141] Y. Li, D. Zha, P. K. Venugopal, N. Zou, X. Hu, Pyodds: An end-to-end outlier detection system with automated machine learning. doi:2020.
URL http://arxiv.org/pdf/2003.05602v1

[142] N. Stojanovic, M. Dinic, L. Sotjanovic, A data-driven approach for multivariate contextualized anomaly detection: industry, in: J.-Y. Nie, Z. Obradovic, T. Suzumura, R. Ghosh, R. Nambiar, C. Wang (Eds.), 2017 IEEE International Conference on Big Data, IEEE, Piscataway, NJ, 2017, pp. 1560–1570.

[143] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, Q. Zhang, Time-series anomaly detection service at microsoft, in: A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, G. Karypis (Eds.), Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, New York, NY, USA, 2019, pp. 3009–3017. doi:10.1145/3292500.3330680.

[144] T. Raeder, O. Stitelman, B. Dalessandro, C. Perlich, F. Provost, Design principles of massive, robust prediction systems, in: Q. Yang, D. Agarwal, J. Pei (Eds.), Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12, ACM Press, New York, New York, USA, 2012, p. 1357. doi:10.1145/2339530.2339740.

[145] T. Rakthanmanon, E. Keogh, Fast shapelets: A scalable algorithm for discovering time series shapelets, in: J. Ghosh, Z. Obradovic, J. Dy, Z.-H. Zhou, C. Kamath, S. Parthasarathy (Eds.), Proceedings of the 2013 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013, pp. 668–676. doi:10.1137/1.9781611972832.74.

[146] L. Ye, E. Keogh, Time series shapelets: a novel technique that allows accurate, interpretable and fast classification, Data mining and knowledge discovery 22 (1-2) (2011) 149–182.

[147] S. Imani, F. Madrid, W. Ding, S. Crouter, E. Keogh, Matrix profile xiii: Time series snippets: A new primitive for time series data mining, in: 2018 IEEE International Conference on Big Knowledge (ICBK), IEEE, 2018.

[148] L. Beggel, B. X. Kausler, M. Schiegg, M. Pfeiffer, B. Bischl, Time series anomaly detection based on shapelet learning, Computational Statistics 34 (3) (2019) 945–976. doi:10.1007/s00180-018-0824-9.

[149] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, E. Keogh, Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016, pp. 1317–1322. doi:10.1109/ICDM.2016.0179.

[150] A. Dau, D. Silva, F. Petitjean, G. Forestier, A. Bagnall, E. Keogh, Judicious setting of dynamic time warping's window width allows more accurate classification of time series, in: 2017 IEEE International Conference on Big Data, IEEE, 2017, pp. 917–922.

[151] E. Keogh, Indexing and mining time series data, in: S. Sekhar, H. Xiong (Eds.), Encyclopedia of GIS, Springer, Boston, Massachusetts, USA, 2008, pp. 493–497.

[152] E. Keogh, M. Pazzani, Relevance feedback retrieval of time series data, in: F. Gey, M. Hearst, R. Tong (Eds.), Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA, 1999, pp. 183–190.

[153] J. Lin, E. Keogh, S. Lonardi, P. Patel, Finding motifs in time series (2002).

[154] C.-C. M. Yeh, E. van Herle, E. Keogh, Matrix profile iii: The matrix profile allows visualization of salient subsequences in massive time series, in: 2016 IEEE 16th International Conference on Data Mining (ICDM), IEEE, 2016.

[155] J. Klemming, Ucr dtw (2018).
URL https://github.com/klon/ucrdtw

[156] S. Gharghabi, S. Imani, A. Bagnall, A. Darvishzadeh, E. Keogh, An ultra-fast time series distance measure to allow data mining in more complex real-world deployments, Data Mining and Knowledge Discovery 34 (4) (2020) 1104–1135. doi:10.1007/s10618-020-00695-8.

[157] K. Kamgar, S. Gharghabi, E. Keogh, Matrix profile xv: Exploiting time series consensus motifs to find structure in time series sets (2019).

[158] M. Linardi, Y. Zhu, T. Palpanas, E. Keogh, Matrix profile x: Valmod - scalable discovery of variable-length motifs in data series, in: G. Das, C. Jermaine, P. Bernstein (Eds.), Proceedings of the 2018 International Conference on Management of Data, ACM, New York, NY, USA, 2018, pp. 1053–1066. doi:10.1145/3183713.3183744.

[159] Y. N. Silva, S. S. Pearson, J. Chon, R. Roberts, Similarity joins: Their implementation and interactions with other database operators, Information Systems 52 (2015) 149–162. doi:10.1016/j.is.2015.01.008.

[160] W. Mann, N. Augsten, P. Bouros, An empirical evaluation of set similarity join techniques, Proceedings of the VLDB Endowment (9) (2016) 636–647.

[161] C. Myers, L. Rabiner, A. Rosenberg, Performance tradeoffs in dynamic time warping algorithms for isolated word recognition, IEEE Transactions on Acoustics, Speech, and Signal Processing 28 (1980) 623–635.

[162] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, IEEE Transactions on Acoustics, Speech, and Signal Processing 26 (1) (1978) 43–49. doi:10.1109/TASSP.1978.1163055.

[163] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Searching and mining trillions of time series subsequences under dynamic time warping, in: Q. Yang (Ed.), Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, 2012, pp. 262–270.

[164] J. Paparrizos, L. Gravano, k-shape: Efficient and accurate clustering of time series, ACM SIGMOD Record 45 (2016) 69–76. doi:10.1145/2949741.2949758.

[165] J. Gu, X. Jin, A Simple Approximation for Dynamic Time Warping Search in Large Time Series Database, 2006. doi:10.1007/11875581{\textunderscore}101.

[166] J. Alon, V. Athitsos, Q. Yuan, S. Sclaroff, A unified framework for gesture recognition and spatiotemporal gesture segmentation, IEEE transactions on pattern analysis and machine intelligence 31 (9) (2009) 1685–1699. doi:10.1109/TPAMI.2008.203.

[167] T. Schlegl, S. Schlegl, D. Tomaselli, N. West, J. Deuse, Adaptive similarity search for the retrieval of rare events from large time series databases, Journal of Advanced Engineering Informatics 52 (C) (2022).

[168] E. Keogh, Exact indexing of dynamic time warping, Knowledge and Information Systems 62 (12) (2005) 358–386.

[169] L. Rabiner, B.-H. Juang, Fundamentals of Speech Recognition, Prentice-Hall, Inc, USA, 1993.

[170] S. Gharghabi, Y. Ding, C.-C. M. Yeh, K. Kamgar, L. Ulanova, E. Keogh, Matrix profile viii: Domain agnostic online semantic segmentation at superhuman performance levels, in: 17th International Conference on Data Mining (ICDM), IEEE, 2017, pp. 117–126.

[171] S. Imani, F. Madrid, W. Ding, S. E. Crouter, E. Keogh, Introducing time series snippets: a new primitive for summarizing long time series, Data Mining and Knowledge Discovery 34 (6) (2020) 1713–1743. doi:10.1007/s10618-020-00702-y.

[172] M. Arul, A. Kareem, Applications of shapelet transform to time series classification of earthquake, wind and wave data, Engineering Structures 228 (2021) 111564. doi:10.1016/j.engstruct.2020.111564.

[173] C. Ji, S. Liu, C. Yang, L. Pan, L. Wu, X. Meng, A shapelet selection algorithm for time series classification: New directions, Procedia Computer Science 129 (2018) 461–467. doi:10.1016/j.procs.2018.03.025.

[174] J. Grabocka, N. Schilling, M. Wistuba, L. Schmidt-Thieme, Learning time-series shapelets, in: S. Macskassy, C. Perlich, J. Leskovec, W. Wang, R. Ghani (Eds.), 20th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, New York, NY, USA, 2017, pp. 392–401.

[175] C. Shannon, The mathematical theory of communication, Bell System Technical Journal 27 (3) (1948) 379–423.

[176] L. Ye, E. Keogh, Time series shapelets: A new primitive for data mining, in: J. Elder (Ed.), Proceedings of the 15th ACM KDD International Conference on Knowledge Discovery & Data Mining, ACM, New York, NY, United States, 2009, pp. 947–956.

[177] A. Mueen, E. Keogh, N. Young, Logical-shapelets: An expressive primitive for time series classification, in: Proceedings of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, 2011.

[178] J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation, Data Mining and Knowledge Discovery 28 (4) (2014) 851–881. doi:10.1007/s10618-013-0322-1.

[179] Q. Cai, C. Zhang, C. Peng, Analysis of classification margin for classification accuracy with applications, Neurocomputing 72 (7-9) (2009) 1960–1968. doi:10.1016/j.neucom.2008.03.015.

[180] J. Grabocka, M. Wistuba, L. Schmidt-Thieme, Fast classification of univariate and multivariate time series through shapelet discovery, Knowledge and Information Systems 49 (2) (2016) 429–454. doi:10.1007/s10115-015-0905-9.

[181] R. Gilad-Bachrach, A. Navot, N. Tishby, Margin based feature selection - theory and algorithms, in: C. Brodley (Ed.), Twenty-first international conference on Machine learning - ICML '04, ACM Press, New York, New York, USA, 2004, p. 43.

[182] C. Burges, A tutorial on support vector machines for pattern recognition, Data mining and knowledge discovery 2 (1998) 121–167.

[183] Q. Yan, Y. Cao, Optimizing shapelets quality measure for imbalanced time series classification, Applied Intelligence 50 (2) (2020) 519–536. doi:10.1007/s10489-019-01535-z.

[184] D. Chieslak, N. Chawla, Learning decision trees for unbalanced data, Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2008) 241–256.

[185] B. Braunmueller, M. Ester, H.-P. Kriegel, J. Sander, Efficiently supporting multiple similarity queries for mining in metric databases, in: Proceedings of 16th International Conference on Data Engineering, IEEE, 2000, pp. 256–267.

[186] T. Seidl, H.-P. Kriegel, Adaptable similarity search in large image databases, in: R. Veltkamp, H. Burkhardt, H.-P. Kriegel (Eds.), State-of-the-Art in Content-Based Image and Video Retrieval, Springer, 2001, pp. 297–317.

[187] G. Marchionini, Exploratory search: From finding to understanding, Communications of the ACM 49 (4) (2006) 41–46.

[188] R. W. White, R. A. Roth, Exploratory search: Beyond the query-response paradigm, Synthesis Lectures on Information Concepts, Retrieval, and Services 1 (1) (2009) 1–98. doi:10.2200/S00174ED1V01Y200901ICR003.

[189] X. Jin, M. Sloan, J. Wang, Interactive Exploratory Search for Multi Page Search Results, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, 2013.

[190] W.-T. Fu, T. Kannampallil, R. Kang, Facilitating exploratory search by model-based navigational cues, in: Proceedings of the 27th Annual CHI Conference on Human Factors in Computing Systems, 2009, pp. 199–208.

[191] M. Patella, P. Ciaccia, Approximate similarity search: A multi-faceted problem, Journal of Discrete Algorithms 7 (1) (2009) 36–48. doi:10.1016/j.jda.2008.09.014.

[192] C. Manning, P. Raghavan, H. Schuetze, Introduction to Information Retrieval, Cambridge University Press, 2009.

[193] Y. Lv, C. Zhai, Adaptive relevance feedback in information retrieval, in: D. Cheung (Ed.), Proceedings of the ACM 18th International Conference on Information and Knowledge Management, ACM, New York, NY, USA, 2009.

[194] O. Kurland, The cluster hypothesis in information retrieval, Advances in Information Retrieval (2014).

[195] N. Monty, Deep blue's contribution to ai, Annals of Mathematics and Artificial Intelligence 28 (2000) 27–30.

[196] S. D. Holcomb, W. K. Porter, S. V. Ault, G. Mao, J. Wang, Overview on deepmind and its alphago zero ai, in: Proceedings of the 2018 International Conference on Big Data and Education, ACM, New York, NY, USA, 2018, pp. 67–71. doi:10.1145/3206157.3206174.

[197] Y. Bengio, Y. LeCun, G. Hinton, Deep learning for ai, Communications of the ACM 64 (7) (2021) 58–65. doi:10.1145/3448250.

[198] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.

[199] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowics, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuter, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, Wattenberg, Martin, Wicke, Martin, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems.

[200] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, Advances in Neural Information Processing Systems 32 (2019) 8024–8035.

[201] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. López García, I. Heredia, P. Malík, L. Hluchý, Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey, Artificial Intelligence Review 52 (1) (2019) 77–124. doi:10.1007/s10462-018-09679-z.

[202] K. Hornik, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.

[203] Y. LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, Lawrence D. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in neural information processing systems, Vol. 1989, Morgan Kaufmann Publishers, 1989.

[204] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: A survey (2019).

[205] V. Strobel, G. Saponaro, T. Spetebroot, T. Vercauteren, J. Wienke, Historic word occurrence in academic papers (2018). doi:10.5281/zenodo.1218409.
URL https://github.com/Pold87/academic-keyword-occurrence

[206] J. Born, D. Beymer, D. Rajan, A. Coy, V. Mukharjee, M. Manica, P. Prasanna, D. Ballah, M. Guindy, D. Shaham, P. Shah, E. Karteris, J. Robertus, M. Gabrani, M. Rosen-Zvi, On the role of artificial intelligence in medical imaging of covid-19, Patterns 2 (2021).

[207] D. Floreano, C. Mattiussi, Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies, MIT Press, Cambrdige, MA, USA, 2008.

[208] S. Ghosh-Dastidar, H. Adeli, Spiking neural networks, International Journal of Neural Systems 19 (4) (2009) 295–308.

[209] W. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics 5 (1943) 115–133.

[210] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychological Review 65 (6) (1958).

[211] C. Murphy, P. Gray, G. Stewart, Verified perceptron convergence theorem, in: T. Shpeisman, J. Gottschlich (Eds.), Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, ACM, New York, NY, USA, 2017, pp. 43–50. doi:10.1145/3088525.3088673.

[212] M. Minsky, S. Papert, Perceptrons: an introduction to computational geometry, 1969.

[213] A. LeNeil, NN-SVG: Publication-Ready Neural Network Architecture Schematics. 4 (33) (2019). [link].
URL https://github.com/alexlenail/NN-SVG

[214] D. Rumelhart, G. Hinton, R. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–537.

[215] N. Gupta, Atrificial neural network, Network and Complex Systems 3 (1) (2013) 24–29.

[216] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems (1989) 300–314.

[217] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning requires rethinking generalization.

[218] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Communications of the ACM 63 (11) (2020) 139–144. doi:10.1145/3422622.

[219] A. Kusiak, Convolutional and generative adversarial neural networks in manufacturing, International Journal of Production Research 58 (5) (2019) 1594–1604.

[220] D. Rumelhart, R. Durbin, R. Golden, Y. Chauvin (Eds.), Backpropagation: Theory, Architectures and Applications: Backpropagation: The Basic Theory, 1st Edition, Psychology Press, 1995.

[221] S. Du, J. Lee, Y. Tian, Poczos, Barnabas, Singh, Aarti, Gradient descent learns one-hidden-layer cnn: Don't be afraig of spurious local minima, in: Proceedings of the 35th International Conference on Machine Learning, 2018.

[222] A. Burkov, The Hundred-Page Machine Learning Book, 2019.

[223] G. E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural computation 18 (7) (2006) 1527–1554. doi:10.1162/neco.2006.18.7.1527.

[224] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Advances in Neural Information Processing Systems 19 (NIPS 2006), MIT Press, 2006.

[225] J. Chung, K. Kastner, L. Dinh, A recurrent latent variable model for sequential data, in: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Sys-tems 28 (NIPS 2015), 2015.

[226] S. Hichreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[227] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, in: NIPS 2014 Workshop on Deep Learning, 2014. doi:NIPS.

[228] D. Park, Y. Hoshi, C. Kemp, A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder, Robotics and Automation Letters (2018) 1544–1551.

[229] T. Schlegl, S. Schlegl, J. Deuse, Scalable anomaly detection in manufacturing systems using an interpretable deep learning approach, in: Procedia CIRP, Elsevier, 2021, pp. 1546–1551.

[230] A. Karpathy, J. Johnson, L. Fei-Fei, Visualizing and understanding recurrent networks, International Conference on Learning Representations (ICLR).

[231] S. Aghabozorgi, A. Seyed Shirkhorshidi, T. Ying Wah, Time-series clustering – a decade review, Information Systems 53 (2015) 16–38.

[232] D. P. Kingma, M. Welling, Auto-encoding variational bayes.
URL http://arxiv.org/pdf/1312.6114v10

[233] J. Langr, V. Bok, GANs in Action, Manning Publications, 2019.

[234] Y. Wang, D. Blei, J. Cunningham, Posterior collapse and latent variable non-identifiability, Advances in Neural Information Processing Systems 34 (NeurIPS 2021) (2021).

[235] J. Lucas, G. Tucker, R. Grosse, M. Norouzi, Understanding posterior collapse in generative latent variable models, in: T. Sainath (Ed.), Seventh International Conference on Learning Representations, 2019.

[236] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine 29 (6) (2012) 141–142.

[237] X. Ding, X. Wu, Y. Wang, Bolt axial stress measurement based on a mode-converted ultrasound method using an electromagnetic acoustic transducer, Ultrasonics 54 (3) (2014) 914–920. doi:10.1016/j.ultras.2013.11.003.

[238] K.-H. Kloos, W. Thomala, Schraubenverbindungen: Grundlagen, Berechnung, Eigenschaften, Handhabung, 5th Edition, Springer-Verlag, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-68470-1.
URL http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10177097

[239] E. Tempelman, H. Shercliff, B. van Ninaber Eyben, Manufacturing and Design: Understanding the principles of how things are made, 1st Edition, Butterworth-Heinemann, 2014.

[240] R. Shoberg, Engineering fundamentals of threaded fastener design and analysis (2021).

[241] T. Erl, Service-Orientated Architecture: Service-Orientated Architecture, 4th Edition, Prentice Hall, 2005.

[242] T. Nakamura, M. Imamura, R. Mercer, E. Keogh, Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives, in: 2020 IEEE International Conference on Data Mining (ICDM), IEEE, 2020, pp. 1190–1195. doi:10.1109/ICDM50108.2020.00147.

[243] A. Halevy, P. Norvig, F. Pereira, The unreasonable effectiveness of data, IEEE Intelligent Systems 24 (2) (2009) 8–12.

[244] T. Schlegl, S. Schlegl, Mlops framework for model lifecycle management (2022).
URL https://github.com/papelero/mlops-framework